Machine Learning Techniques for Detecting Malicious Activities in the Network

Synopsis of the PhD Thesis

Submitted by

Radhika Chapaneri

(SAP ID 70821016005)

Department of Computer Engineering

Supervisor

Dr. Seema Shah

Submitted in partial fulfilment of the requirements of
Doctor of Philosophy (PhD Engineering)

MUKESH PATEL SCHOOL OF TECHNOLOGY, MANAGEMENT & ENGINEERING

SVKM's NARSEE MONJEE INSTITUTE OF MANAGEMENT STUDIES

(Declared as Deemed-to-be University Under Section 3 of the UGC Act, 1956)

V. L. Mehta Road, Vile Parle (West)

MUMBAI - 400056

2022

Contents

1 Introduction 1
1.1 Research Objectives 1
1.2 Contributions 2
2 Related Work 2
3 Comparative Analysis of Datasets 4
3.1 Conventional Datasets 5
3.2 Modern Datasets 5
4 Statistical Decision Method for Detecting Unknown Attacks 7
4.1 Data Pre-processing 8
4.2 GMM Modelling with Adaptive Thresholding 10
4.3 Multi-level Classification 12
4.4 Experimental Results and Analysis 13
5 Combating Imbalance Network Traffic Problem with Regularized WGAN . 15
5.1 GAN based Data Augmentation 16
5.2 WGAN model for data augmentation 18
5.3 Regularized WGAN Models for Data Augmentation 19
5.4 Statistical Validity of Data Augmentation Models 21
5.5 Classification Strategies 23
5.6 Experimental Results and Analysis 23
6 Malicious Traffic Detection using Convolutional Neural Network 27
6.1 Pre-processing of UNSW-NB15 Dataset 27
6.2 Fully-connected Feed-forward Network 27
6.3 One-dimensional Convolutional Neural Network 28
6.4 Experimental Results and Analysis 30
7 Conclusions and Future Scope 31

List of Paper Publications **33**

ABSTRACT

There is a significant increase in malicious traffic over the network in recent years due to the increase in connectivity over the network between end-users. Various vulnerabilities exist in network infrastructure and protocols that are exploited by attackers resulting in disruption of businesses of organizations as well as individual users. Although the methods based on signatures can detect malicious traffic, such methods fail to detect unknown attacks or even variants of known attacks. Thus, machine learning-based methods are required to detect the malicious traffic on the network. However, several problems are still not completely solved with the existing approaches such as detecting unknown attacks over the network, traffic category imbalance, and poor detection performance of low-frequent attacks. In this research work, the following contributions are made: (i) detection of unknown attacks using a multi-level classification approach, (ii) solving the class imbalance problem with the regularized Wasserstein generative adversarial network, and (iii) effective detection of minority traffic class using deep learning approach. Additionally, a detailed comparative analysis of traditional, as well as modern datasets, is performed considering the recent network attacks.

The first contribution is the design of a novel multi-level classification approach for classifying the traffic data samples into known traffic categories as well as detecting unknown attacks. This is achieved with the unsupervised Gaussian mixture modeling approach for learning the statistical patterns of each traffic category followed by an adaptive thresholding mechanism for detecting unknown attacks. The experimental evaluation is performed on the modern benchmark CICIDS2017 dataset and the results demonstrate an improvement in detecting the known as well as the unknown attacks compared to the state-of-the-art methods.

The second contribution is solving the problem of traffic category imbalance. A regularization method based on Wasserstein Generative Adversarial Networks (WGAN) is proposed to obtain the augmented dataset by synthetically generating traffic data samples. These generated samples are validated statistically for quality using five statistical measures and the results show that the proposed augmentation method of Wasserstein GAN with Localized Lipschitz Penalty (WGAN-LLP) obtains high-quality traffic data samples. The experimental evaluation is performed on the benchmark CICIDS2017 dataset with respect to binary and multi-class classification using three different classification strategies: TRTR (train with real, test with real), TSTR (train with synthetic, test with real), and TRTS (train with real, test with synthetic). The results demonstrate that the proposed WGAN-LLP method outperforms the existing techniques since the generated traffic samples are diverse as well as realistic with the binary classification F1 score of 99% and multi-class classification F1 score of 98%.

The third contribution is the design of a deep convolutional neural network architecture for improving the detection performance of traffic attacks that occur less frequently. The experiments are performed on the challenging and complex UNSW-NB15 dataset. The proposed system is shown to outperform the existing classical machine learning as well as shallow neural network methods for detecting low-frequently occurring attacks such as Worms and Shellcode.

1 Introduction

The rapid growth of connectivity between the devices has an immense impact on increasing various sophisticated malicious activities. The vulnerabilities are exploited for stealing valuable information as well as to slow down the network services. The cyber-attacks range from the man-in-the-middle attack to the denial of service attack, which can damage critical infrastructures leading to severe damage and financial losses. Current protection systems are designed on the basis of logging attack signatures/rules and detecting attacks by firing any of these rules but, although they are effective to some extent at identifying existing malicious patterns, any new patterns or even variants of existing activities can hack a victim system. Considering that many malicious activities emerge daily with massive network traffic, there is a pressing need to develop intelligent techniques to tackle these challenges. Recently, several researchers have proposed various machine learning and deep learning models to enhance the attack detection rate. Although promising results have been achieved, the detection rate of minority attack classes is still low due to the imbalanced distribution of network traffic data. Improving the detection rate of known as well as novel/unknown attacks is a crucial problem to be solved for detecting malicious activities in networks.

The problem statement is "to develop effective network anomaly detection methods using machine learning or deep learning techniques for detection of known as well as unknown malicious network traffic." Following challenges are observed in the detection of malicious network traffic: a) huge volume of data: the network traffic is huge and increasing, b) detection of unknown attacks, c) since the network anomalies or attacks are minor instances as compared to benign samples, the class distribution is skewed which imposes the class imbalance problem that adversely affects the detection rate.

1.1 Research Objectives

To address these challenges, the following research objectives and questions are formulated:

RO_1 : To perform a critical analysis of traditional and modern datasets with respect to network scenarios and attacks.
RQ1: Which dataset should be considered for the performance evaluation?

RO_2 : To detect unknown attacks from the network traffic using unsupervised learning.
RQ2: Which statistical models can be used to detect unknown attacks?

RO_3 : To solve the network traffic class imbalance problem by proposing a data augmentation model for improved detection performance.
RQ3: How to generate quality network traffic samples to solve the class imbalance problem?

RO_4 : To determine the statistical validity of the generated network traffic samples.
RQ4: How to measure the validity of the generated samples?

RO_5 : To enhance the detection performance of malicious network traffic using deep learning.
RQ5: Can deep learning architectures be effective in improving the detection rate of malicious network traffic, especially the rare classes?

1.2 Contributions

The contributions of this research work are summarized as follows:

- C_1 : Comparative analysis of datasets, pre-processing and visualization of two recent UNSW-NB15 and CICIDS2017 datasets is performed.
- C_2 : Proposed a novel sample selection algorithm to select the representative samples of a dataset.
- C_3 : Proposed a statistical decision engine with an unsupervised multi-level approach based on Gaussian mixtures with adaptive thresholding to detect known as well as unknown network attacks.
- C_4 : Proposed a data augmentation model to solve the class imbalance problem using regularized Wasserstein generative adversarial networks which generates diverse and realistic augmented samples.
- C_5 : Statistical validity using various parametric and non parametric tests is performed to evaluate the quality of generated network traffic data and detailed experiments are performed using three classification strategies for binary and multi-class classification
- C_6 : Proposed a deep one-dimensional convolutional neural network (CNN) architecture which improved the performance of low frequent attacks (Analysis, Backdoor, Shellcode, and Worms).

2 Related Work

Malicious activities are evolving every day and thus becoming harder to detect. Robust security measures and strong defense strategies are required to deal with various known and zero-day attacks. An Intrusion Detection System (IDS) can be used to monitor both host- and network-based systems [1]. A Host-based IDS (HIDS) monitors the traces of a host by gathering information about events that take place in a computer system. It should be installed on each client to monitor it and log its operating system's behaviours to provide audit trails for recognizing both anomalous and legitimate observations. In contrast, a Network-based IDS (NIDS) monitors a network's traffic to detect attacks occurring in its environment. An IDS methodology can be classified as Misuse based Detection System (MDS) and Anomaly-based Detection System (ADS). The MDS monitors network traffic to match observed behaviours against those on a known blacklist but it cannot detect zero-day or even variants of known ones. In contrast, the anomaly-based techniques learn the normal traffic behaviour of the network and raise an alert whenever an anomalous behaviour is detected. In [2], the authors compared the performance of open-source signature-based systems Snort and Suricata; it was observed that since Suricata has multi-threaded architecture, it can process network traffic at a higher speed than Snort, but it consumes excessive computational resources. A significant disadvantage of such rule-based systems is that they cannot detect unknown malicious network traffic. Various anomaly-based approaches, such as nearest neighbor, clustering, statistical, and information-theoretic, along with their advantages and disadvantages, are discussed in [3]. In the Local Outlier Factor (LOF) approach, an outlier score is assigned to each data point based on distances from the local neighborhood. However, the LOF technique has a drawback in identifying outliers in datasets with varying densities. In clustering-based approaches, the data points that do not belong to any cluster are considered outliers. Since clustering aims to find

clusters, these techniques are not optimized to find outliers. Anomaly-based techniques face the challenge of determining a way of creating a comprehensive profile from diverse legitimate behaviours. The output from anomaly detection is often based on a baseline or threshold which is a condition that discriminates between normal and attack instances [3]. Determining this threshold is one of the significant challenges faced when designing an ADS.

Another issue in designing the detection technique for malicious activity is an imbalance problem. A dataset is imbalanced if there is a disproportion in the distribution of data samples among various classes; this phenomenon is observed in multiple domains such as banking fraud detection, medical diagnosis, etc. In the case of the network security domain, this problem is more prevalent as the amount of attack data is always less than the amount of benign data, and also, not all types of attacks occur frequently. The attacker can focus on the minority attacks if the detection of such attacks is low and thus exploit the network vulnerabilities. In the case of network traffic, typically, the benign or normal samples frequently occur more than the attack samples; this phenomenon leads to the data imbalance problem in any network traffic dataset. The class imbalance problem hinders the classifier performance resulting in low robustness to minority attacks. Hence, it is crucial to address this issue of imbalanced data.

SMOTE and its variant ADASYN (ADAptive SYNthetic sampling) is used in [4] for data generation, but the experimental results showed that this technique did not perform well; thus, the authors suggested using deep learning models to solve this problem. In [5], a combination of SMOTE with cluster centers and nearest neighbors is used; however, the experimentation was performed on older datasets, making the detection challenging as these methods are not suitable for detecting modern network attack patterns. Although SMOTE is extensively used in the literature to solve class imbalance problems, it depends on interpolation to conduct oversampling, resulting in a poor representation of the synthetic samples. In [6], a combination of the K-means algorithm and SMOTE was proposed to remove noise by emphasizing the between-class and within-class imbalance. A combination of SMOTE and Gaussian Mixture Model (GMM) was proposed in [7] where SMOTE was used for oversampling of the minority class, and GMM was used for reduction of the samples by clustering. Undersampling may remove relevant data samples, and oversampling may introduce too much redundant data that can make the model prone to overfitting. A balanced multiclass dataset algorithm with SMOTE was proposed in [8], resulting in improved performance compared to the imbalanced dataset; however, it did not provide the performance evaluation of the minority classes. Although these existing works have improved the detection of malicious network traffic, they have the drawback of not generating high-quality samples.

To handle the problem of imbalanced data, GAN [9] can be beneficial as it can learn the probability distribution of the data and generate new samples from this learned distribution. Since GAN can represent arbitrary complex data, it has been widely used in numerous domains such as computer vision [10], fault diagnosis [11], fraud detection [12], etc. In [13], GAN was used for generating adversarial samples to fool the IDS. An anti-intrusion detection auto-encoder was proposed in [14] for learning the distribution of benign samples and generating features that the attackers can use to create real samples for bypassing the existing IDS. Limited work in the literature has used GAN for solving the imbalanced data for detecting malicious activities in the network. GAN was used in [15] to generate training data samples for solving the imbalance problem; however, the experimentation

was performed on older datasets, and the results are not comparable for recent datasets with modern network traffic. IGAN-IDS was proposed in [16]in which an imbalanced data filter was used to calculate the number of samples needed for minority classes, and the samples were generated using a generative adversarial network; the classification was performed using a convolutional neural network. The quality of the generated samples is a critical factor in improving the detection of malicious network traffic. Although GAN has been used in the existing work to solve the class imbalance problem, GAN is known to suffer from mode collapse and instability issues; also, the statistical significance of the generated samples is not evaluated.

A comparison of shallow and deep neural networks with classical machine learning approaches is done in [17] and deep learning models are shown to perform relatively better. In this and other related works, researchers have primarily performed binary classification and evaluated on outdated datasets such as NSL-KDD. It is crucial to evaluate the performance using recent datasets containing modern network traffic and attacks so that the security administrators can take proper actions to mitigate those attacks. In [18], numeric features of the UNSWNB15 dataset were converted to binary and subsequently to 8×8 grayscale image format and modeled using a 2D CNN architecture due to which a high performance was observed for the Normal, Generic, and Exploit classes but the detection rate is almost zero for other classes. It is important to investigate the performance of deep learning models to deal with the high-dimensional nonlinear network data which can improve minority class performance.

To summarize, we highlight the shortcomings of the current detection methods: (i) detection of unknown attacks, (ii) several existing works focus only on binary classification, which is insufficient as different attacks need to be handled separately, (iii) for multi-class classification, the individual class performance is typically not reported due to which the overall accuracy conceals the actual performance of minority classes [19], (iv) the dataset used for testing the system is outdated and does not reflect an actual performance for the modern network traffic, and (v) they do not address the class imbalance problem as insufficient data limits the training of machine learning model.

3 Comparative Analysis of Datasets

Benchmark datasets are required for evaluating the performance of the learned model to detect malicious activities on the network. Datasets can also be used to validate novel methods as well as repeat experiments of existing methods. If the data is not adequately available (i.e. it is scarce), then the detection of malicious network activity can be hampered and this can result in inaccurate evaluation and improper deployment in the network. A major challenge in obtaining the data is the issue of privacy since the network traffic cannot be publicly shared with everyone. Also, the data could undergo anonymization to hide confidential information. Moreover, the data should reflect the modern network traffic trends and should not be obsolete; also, the data should have statistical properties which can help the learning model to generalize the patterns. Generally, the datasets for detecting malicious network activities have the following feature categories: (i) **basic features**: these include the features corresponding to the TCP/IP network connections, (ii) **features based on timestamp**: these include the features that model the temporal properties of the network traffic, and (iii) **features based on content**: these include the

features that are obtained with the domain knowledge so that the malicious/suspicious activity can be captured.

3.1 Conventional Datasets

The first benchmark dataset was KDD99 [21] which was created with an extensive testbed network to simulate of four groups: DOS, U2R, R2L and Probe. KDD99 consists of 41 features and a label to specify either normal or attack (and type of attack). Even though widely used, this dataset has inherent flaws as pointed out in [21]. To avoid these limitations, NSL-KDD dataset was created having the following advantages [21]: a) Redundant records were removed to decrease classifier bias, b) Duplicate records were removed, c) New Train and Test sets were created containing reasonable amount of samples for full set training and testing, and d) Number of samples are inversely proportion to difficulty level of the particular attack category.

3.2 Modern Datasets

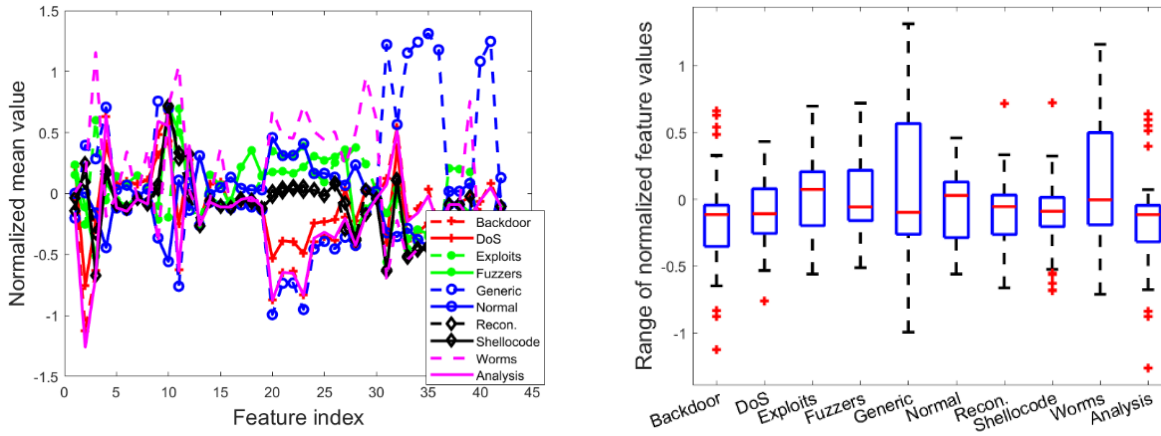
Since the complexity of network and types of attacks have changed over time, the NSL-KDD dataset is therefore not relevant for current evaluation of modern network scenarios. Hence, efforts are geared towards creating benchmark datasets that can capture modern attack traces either via simulation or real-time network traffic. Recent datasets are discussed followed by a comparative analysis of three specific datasets with NSL-KDD in Table 1.

Table 1: Comparison of benchmark and recent IDS datasets

Parameters	**NSL-KDD**	**UNSW-NB15**	**TUIDS**	**CICIDS2017**
Year	2009	2015	2015	2017
Modern network attacks	No	Yes	Yes	Yes
Type of network traffic	Simulated	Simulated	Real	Real
Attack traffic generation	Simulated	IXIA perfect storm tool	Modern tools in testbed network	Modern tools using Kali Linux
Duration	7 weeks	16 hours	7 days	5 days
Feature extraction tools	BroIDS	BroIDS, Argus, C#	Distributed framework	CICFlowmeter
Feature categories	Basic, content, time, connection	Basic, content, time, connection	Basic, content, time, connection	Basic, content, time, connection
Number of features	41	47	50	80
Number of attack classes	4	9	2	10
Publicly available	Yes	Yes	No	Yes

- **UNSW-NB15** [22]: This dataset represents modern attacks and IXIA perfect storm tool was used to create modern network scenarios and nine traffic attack classes. Since it contains information about new attack types updated continuously from CVE (dictionary of publicly known information security vulnerabilities), this dataset

captures modern network intrusion characteristics. From the network traces, 47 features were extracted that include basic, content, time, connection, and additional features. UNSW-NB15 is a complex dataset as it represents modern network and attack traffic and can be used for reliable evaluation. Fig. 1 (a) illustrates analysis



(a) Normalized features (b) Boxplot showing feature variations

Fig. 1: Feature analysis of UNSW-NB15 dataset

of feature values of this dataset where all features are normalized via z-score. Using the boxplot analysis of Fig. 1 (b), we observe that there is significant variation of feature values among the 10 categories which will result in better discrimination for classifiers.

- **TUIDS** [23]: A testbed network was developed at Tezpur University consisting of 5 different networks by configuring 250 hosts across several VLANs, multiple switches, wireless controllers and routers. Real-time network traffic was generated based on daily activities of students, faculties, system administrators, etc., and attack traffic was generated by launching attacks in testbed network using modern tools such as targa, nmap, brutessh, etc. Network traffic of 7 days was captured using Libpcap, Gulp and fast distributed framework was used to extract 50 features consisting of basic (25), content (5), time (10) and connection (10) based traffic features. Since this dataset is not publicly available, we could not do feature analysis of this dataset.

- There are few other datasets such as CAIDA, LBNL and ISCI but they have the problem of anonymity and payload is completely removed because of which there is less utility of these datasets to researchers. KYOTO dataset is made using Honeypots consisting of email and DNS traffic only. IRSC dataset contains both simulated and real attack data but it is not publicly available.

- **CICIDS2017**: This dataset [24] consists of realistic modern network traffic data with the attack classes as per the recent McAfee reports. It contains various protocols, including HTTPS, HTTP, FTP, and SSH, with labeled data. Besides the benign samples, the attack classes of the CICIDS2017 dataset are: (i) DoS: DDoS, Slowloris, Slowhttptest, Hulk, GoldenEye, Heartbleed, (ii) PortScan, (iii) Bot, (iv) BruteForce: FTP-Patator, SSH-Patator, (v) WebAttacks. The balance ratio (BR) and the volume (V) given by Eq. (1) of the CICIDS2017 dataset is depicted in Table 2. It can be

observed that most of the attack classes contribute an insignificant volume in the dataset relative to the Benign class.

$$BR(c) = \frac{\#\,\text{attack samples of class } c}{\#\,\text{benign samples}}, V(c) = \frac{\#\,\text{attack samples of class } c}{\#\,\text{total samples}} \quad (1)$$

Table 2: Balance ratio and Volume of CICIDS2017 dataset

Class	**# samples**	$BR\,(\%)$	$V\,(\%)$	**Class**	**# samples**	$BR\,(\%)$	$V\,(\%)$
Benign	2,273,097	-	80.30	Hulk	231,073	10.17	8.16
Portscan	158,930	6.99	5.61	DDoS	128,027	5.63	4.52
Goldeneye	10,293	0.45	0.36	FTP	7,938	0.35	0.28
SSH	5,897	0.26	0.21	Slowloris	5,796	0.25	0.20
Slowhttp	5,499	0.24	0.19	WebAttacks	2,180	0.10	0.08
Bot	1,966	0.09	0.07				

From our analysis of Table 1, we observe that the recent two datasets UNSW-NB15 and CICIDS2017 satisfies many guidelines set and thus we use them in this work for performance evaluation.

4 Statistical Decision Method for Detecting Unknown Attacks

Existing supervised machine learning methods cannot detect unknown or novel network traffic attacks. To solve this problem, we propose a multi-level statistical decision method that not only classifies the network traffic into several known classes but can also detect novel attacks. A fundamental assumption is that the benign or normal traffic data share common patterns, while the anomalous data in the form of the attack deviate from such patterns. The unsupervised method of Gaussian Mixture Modeling (GMM) is used to learn the statistical characteristics of each network traffic category, and an adaptive thresholding technique based on the interquartile range is used to identify novel attacks. We also propose a sample selection algorithm to select the representative samples from each traffic category automatically to reduce the processing complexity. During the training phase, GMM models are learned for all known network traffic categories and evaluated using the metric of F1 score. The models are then arranged sequentially as per the descending F1 scores. In the testing phase, the traffic category of test data is predicted using proposed multi-level classification method that can also detect unknown attacks. The proposed work is evaluated on the benchmark CICIDS2017 dataset that includes modern network traffic patterns. The experimental results show a significant improvement relative to the state-of-the-art techniques for detecting unknown attacks and classifying multiple network traffic attacks.

Fig. 2 shows the proposed framework for anomaly detection of network traffic using the multi-level Gaussian mixture modeling approach. It consists of several steps for data pre-processing followed by learning the Gaussian mixture model for each class. A multi-level classification technique is proposed based on the learned GMM models for

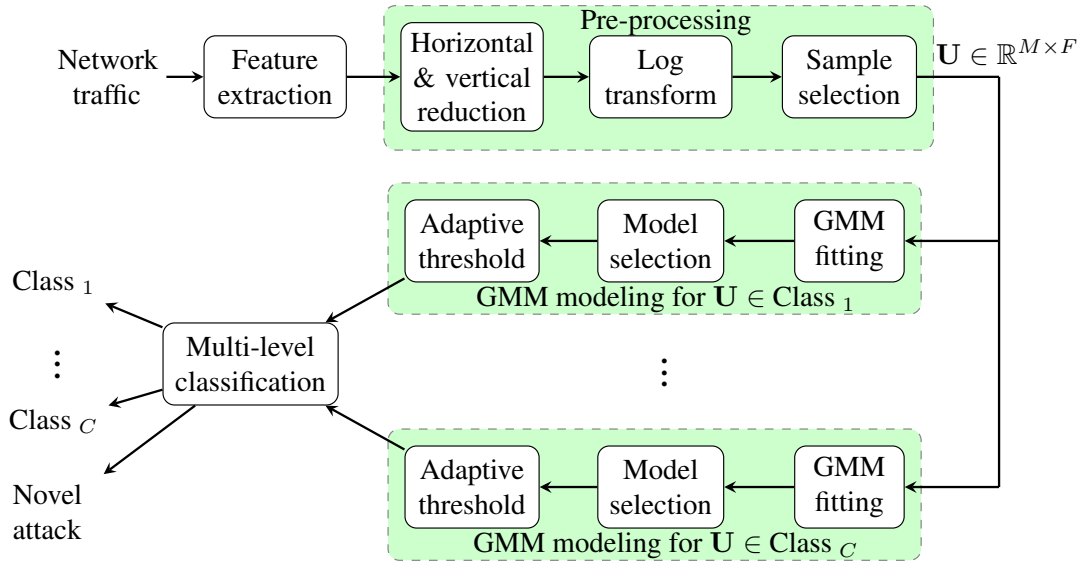


Fig. 2: Proposed framework

classifying the input network traffic as either benign or a specific attack category. The multi-level technique can also detect any novel attack categories not already learned by the system.

4.1 Data Pre-processing

For data pre-processing, the data is initially cleaned via horizontal and vertical data reduction, i.e. certain data samples having NaN values are removed; the FwdHeaderLength feature is mentioned twice in the CICIDS2017 dataset and hence the repeating column is deleted; the features having all zero values are dropped: a) BwdPshFlags, b) FwdURGFlags, c) BwdURGFlags, d) CWEFlagCount, e) FwdAvgBytesBulk, f) FwdAvgPacketsBulk, g) FwdAvgBulkRate, h) BwdAvgBulkRate, i) BwdAvgPacketBulk, and j) BwdAvgBulkrate; to avoid the redundancy, the packet length variance feature is removed due to the presence of packet length std deviation feature; the irrelevant data samples having negative values for the following features are removed: a) FlowDuration, b) FlowBytes, c) FlowPackets/s, d) FlowIATMean, e) BwdIAT, f) FwdHeaderLength and g) BwdHeaderLength. These features are required to have non-negative values and thus these irrelevant samples are removed. The features are then transformed to log space since several feature values span varying orders of magnitude, for example, the range of FlowDuration is from 900 to 1,600,000. All feature values are transformed into the log domain using $x \leftarrow \log(x+1)$ thus reducing their scale and aiding the learning model to work with smaller feature values.

4.1.1 Sample Selection

Considering the distribution of attacks in the CICIDS2017 dataset as shown in Table 3, there are more benign samples relative to the malicious samples. Also, the distribution of attacks is highly imbalanced, for example, the DoS attack is almost half of all attacks and the Portscan attack is one-third of all attacks as they cause too many packets flows during the attack.

Algorithm 1 Representative Sample Selection

Input: $\mathbf{U}^{'}$ $(M_i^{'} \times F)$, distance threshold δ
Output: Representative selected samples $\mathbf{U}$ $(M_i \times F)$

1: $\mathbf{r}_1 = \mathbf{U}^{'}(1,:)$, $M_i = 1$ ▷ first cluster centroid
2: **for** $l = 2, \ldots, M_i^{'}$ **do**
3: $\quad \mathbf{v} = \mathbf{U}^{'}(l,:)$ ▷ read the next sample
4: $\quad$ **if** $\exists \mathbf{r}_q \in \{\mathbf{r}_h : 1 \leq h \leq M_i\},\ d(\mathbf{v}, \mathbf{r}_q) < \delta$ **then**
5: $\quad\quad$ put $\mathbf{v}$ in q^{th} cluster and update the cluster centroid $\mathbf{r}_q$
6: $\quad$ **else**
7: $\quad\quad$ $M_i = M_i + 1$, $\mathbf{r}_{M_i} = \mathbf{v}$ ▷ update number of clusters and create a new cluster
8: $\mathbf{U} = \left[\mathbf{r}_1, \ldots, \mathbf{r}_{M_i}\right]^\top$ ▷ representative samples

To select the most representative samples for each category, we propose a sample selection algorithm given by Algorithm 1 taking input as $\mathbf{U}'$ consisting of M' data samples resulting from horizontal reduction with F features and a distance threshold δ. The threshold δ is a hyper-parameter that can be obtained as the median of all distance pairs for a particular category. The first cluster centroid $\mathbf{r}_1$ is chosen as the first instance of the dataset. Each subsequent sample of the dataset is assigned to $\mathbf{v}$ whose distance d is computed from the clusters generated so far in line 5. If the distance d is less than the threshold δ with respect to the q^{th} cluster, then this sample $\mathbf{v}$ is included in the q^{th} cluster and the centroid of cluster $\mathbf{r}_q$ is updated in a manner similar to the method adopted in K-means clustering. Otherwise, this sample defines a new cluster $\mathbf{r}_{M_i}$ where M_i is the total number of resulting clusters. The representative samples $\mathbf{U} \in \mathbb{R}^{M_i \times F}$ are thus the cluster centroids $\mathbf{r}_1, \ldots, \mathbf{r}_{M_i}$. This algorithm is repeated for each category of data, i.e. Benign, DoS, etc. Table 3 shows the number of samples before sample selection as well as the number of representative samples after the sample selection algorithm.

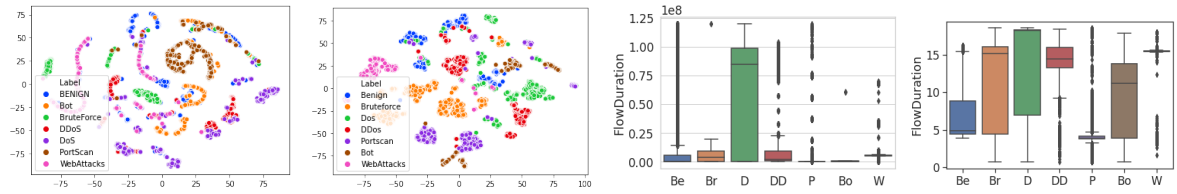
Table 3: Representative samples of CICIDS 2017 dataset after sample selection

Class	N_O	N_A	N_S	**Class**	N_O	N_A	N_S
Benign	529,918	231,187	4,195	Bot	1,966	1,956	1,956
Bruteforce	13,835	13,826	3,998	DDos	128,027	128,006	4,415
DoS	252,672	251,555	4,954	Portscan	158,930	158,797	3,081
WebAttacks	2,180	2,180	2,180	**Total**	**1,087,528**	**787,507**	**24,779**

N_O: # original samples, N_A: after preprocessing, N_S: after selection

4.1.2 Data Visualization

The t-SNE scatter plot of CICIDS2017 dataset projected onto two dimensions without and with pre-processing is shown in Fig. 3 (a) & (b). It can be observed that there is considerable overlap between various traffic classes without pre-processing, whereas the proposed pre-processing results in almost distinct clusters and thus discriminative features which can reduce the effort of learning models. Also, The boxplot of two features across the traffic categories is shown in Fig. 3 (c) & (d), where the advantage of proposed data pre-processing steps can be observed as the feature representation after pre-processing is more discriminative for machine learning models to learn.



(a) Before pre-processing (b) After pre-processing (c) Before pre-processing (d) After pre-processing

Fig. 3: t-SNE visualization (left) and Boxplot visualization (right) for the CICIDS2017 dataset

4.2 GMM Modelling with Adaptive Thresholding

4.2.1 Parameter Estimation of GMM

Since the data cannot be represented appropriately with only one Gaussian as per the HZ test and QQ plots, H mixtures of Gaussians are used given by Eq. (2) with model parameters $\{\tau_h, \boldsymbol{\nu}_h, \boldsymbol{\Gamma}_h\}_{h=1}^{H}$ indicating the weight, mean and covariance of the h^{th} latent variable (or mixture), and $\mathbf{u} \in \mathbb{R}^F$ is the data sample having F features. In GMM, $\mathbf{q}$ is a H-dimensional binary random variable having a one-hot representation with $q_h \in \{0, 1\}$ and $\sum_h q_h = 1$.

$$\mathbb{P}(\mathbf{u}) = \sum_{\mathbf{q}} \mathbb{P}(\mathbf{q})\mathbb{P}(\mathbf{u}|\mathbf{q}) = \sum_{\mathbf{q}} \prod_{i=1}^{H} \left(\tau_i \mathcal{N}\left(\mathbf{u}|\boldsymbol{\nu}_i, \boldsymbol{\Gamma}_i\right)\right)^{q_i} \tag{2}$$

The goal of GMM is to estimate the model parameters using the Maximum Likelihood Estimation (MLE) method for which the EM (Expectation Maximization) algorithm is used. The log-likelihood function is given by Eq. (3) assuming that the data points $\mathbf{u}_m$ are sampled independently.

$$\mathcal{L} = \ln p(\mathbf{U}|\boldsymbol{\tau}, \boldsymbol{\nu}, \boldsymbol{\Gamma}) = \sum_{m=1}^{M_i} \ln \left\{ \sum_{h=1}^{H} \tau_h \mathcal{N}\left(\mathbf{u}_m|\boldsymbol{\nu}_h, \boldsymbol{\Gamma}_h\right) \right\} \tag{3}$$

The training algorithm of GMM modeling is given by Algorithm 2, which is executed for each class of the dataset, using the update equations given by Eq. (4) and (5).

$$\varphi_{mh} = \frac{\tau_h \mathcal{N}\left(\mathbf{u}_m|\boldsymbol{\nu}_h, \boldsymbol{\Gamma}_h\right)}{\sum_{i=1}^{H} \tau_i \mathcal{N}\left(\mathbf{u}_m|\boldsymbol{\nu}_i, \boldsymbol{\Gamma}_i\right)}, \quad \tau_h = \frac{1}{M} \sum_{m=1}^{M} \varphi_{mh} \tag{4}$$

$$\boldsymbol{\nu}_h = \frac{\sum_{m=1}^{M} \varphi_{mh} \mathbf{u}_m}{\sum_{m=1}^{M} \varphi_{mh}}, \quad \boldsymbol{\Gamma}_h = \frac{\sum_{m=1}^{M} \varphi_{mh} (\mathbf{u}_m - \boldsymbol{\nu}_h)(\mathbf{u}_m - \boldsymbol{\nu}_h)^T}{\sum_{m=1}^{M} \varphi_{mh}} \tag{5}$$

After the EM algorithm converges, the log likelihood value is computed for each training sample and stored in *scores* from which the first (Q_1) and third (Q_3) quartiles are computed. These statistics are used during the testing phase for adaptive thresholding using the inter-quartile range $IQR = Q_3 - Q_1$. Any value lower than $Q_1 - 1.5 \times IQR$ or more than $Q_3 + 1.5 \times IQR$ will have a very less probability hence can be considered to be anomalous with respect to the given distribution.

Algorithm 2 Training - Statistical Anomaly Detection using GMM

Input: $\mathbf{U}\,(M_i \times F)$, H (number of clusters)
Output: $p : \{\boldsymbol{\tau}, \boldsymbol{\nu}, \boldsymbol{\Gamma}, Q_1, Q_3\}$ (profile)

1: Initialize $\tau_h = \frac{1}{H}$, $\boldsymbol{\nu}_h$ as K-means center $(1 \times F)$, $\boldsymbol{\Gamma}_h$ as covariance of the data $(F \times F)$, $\forall h = 1, \ldots, H$, and $\mathcal{L}^{(0)} = -\inf$
2: **E-step**: Compute the posterior probabilities φ_{mh}, $\forall h = 1, \ldots, H$
3: **M-step**: Update the GMM parameters $\tau_h, \boldsymbol{\nu}_h, \boldsymbol{\Gamma}_h$, $\forall h = 1, \ldots, H$
4: Compute the log likelihood $\mathcal{L} = \ln p(\mathbf{U}|\boldsymbol{\tau}, \boldsymbol{\nu}, \boldsymbol{\Gamma})$. If $\mathcal{L}$ not converged, return to Step 3.
5: Obtain log likelihood scores w for every training sample.
6: Compute $Q_1 = 25$ percentile of w and $Q_3 = 75$ percentile of w.

4.2.2 Model Selection

Since the choice of H in GMM is a hyper-parameter, we determine the optimal value of H using model selection with Bayesian Information Criterion (BIC) measure given by Eq. (6), where for any model $\mathcal{M}$, $\mathcal{L}(\mathcal{M})$ represents the log-likelihood of data with respect to $\mathcal{M}$, M is the number of data samples and $g(\mathcal{M})$ is the number of model parameters.

$$BIC(\mathcal{M}|\mathcal{D}) = -2 \times \mathcal{L}(\mathcal{M}|\mathcal{D}) + \ln M \times g(\mathcal{M}) \tag{6}$$

For each class, various values of $H = \{2, 4, 8, 16, 32, 64, 128, 256, 512\}$ is used for training the GMM model, and the model resulting in the lowest BIC value is retained as the optimal model since the BIC measure penalizes models with a high number of clusters. An illustration of the model selection using BIC is shown in Fig. 4 for the benign category.

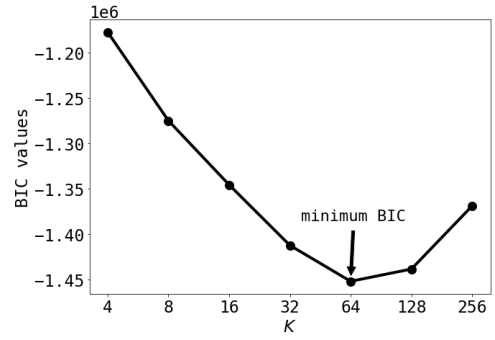


Fig. 4: BIC plot for Benign class

4.2.3 Testing Phase

In the testing phase as shown in Algorithm 3, data samples of all classes are combined together resulting in $\mathbf{U}_t \in \mathbb{R}^{M \times F}$ and the ground truth for each test sample is set as follows: $t_{true} = 1$ if the sample belongs to the desired category, else $t_{true} = 0$. The IQR thresholding technique is used to determine whether the data sample belongs to a particular class or not.

For each test sample, the log-likelihood score is computed and compared with the IQR, calculated as $IQR = Q_3 - Q_1$. The IQR criterion is useful since extreme values has less influence on it as it limits the range to middle 50% of the score values. The values for $Q_1 - 1.5 \times IQR$ and $Q_3 + 1.5 \times IQR$ are the thresholds that are used to determine the

Algorithm 3 Testing - Statistical Anomaly Detection using GMM

Input: $\mathbf{U}_t\,(M \times F)$ (all samples), $profile : \{\boldsymbol{\tau}, \boldsymbol{\nu}, \boldsymbol{\Gamma}, Q_1, Q_3\}$
Output: Decision label for each sample

1: Compute $IQR = Q_3 - Q_1$
2: Compute $l_b = Q_1 - 1.5 \times IQR$ and $u_b = Q_3 + 1.5 \times IQR$
3: **for** each sample $\mathbf{u}_m \in \{\mathbf{U}_t\}_{m=1}^{M}$ **do**
4: Compute the log likelihood score $w(m) = \ln \mathbb{P}(\mathbf{u}_m|\boldsymbol{\tau}, \boldsymbol{\nu}, \boldsymbol{\Gamma})$
5: **if** $w(m) < l_b$ or $w(m) > u_b$ **then**
6: $label(m)$ = 'Anomaly'
7: **else**
8: $label(m)$ = 'Benign'

outlier values. The decision is taken in lines 4 to 8 using the IQR threshold to assign Benign or Anomaly label; to the specific data sample, assuming that the GMM profile of the Benign class profile is given as the input, but this applies in general to any class.

4.3 Multi-level Classification

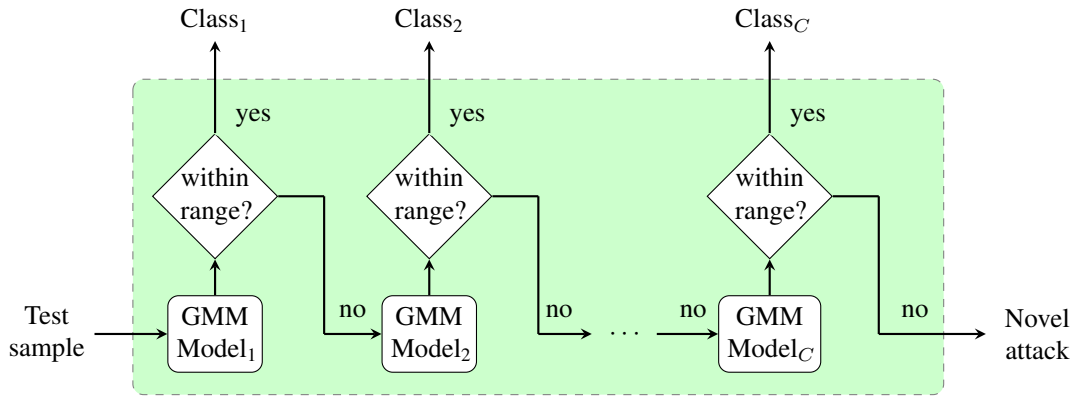


Fig. 5: Proposed multi-level classification framework

To detect the specific class of network traffic data samples as well as to detect novel attacks (zero-day attack), a multi-level classification approach is proposed as shown in Fig. 5. Based on the trained GMM models for each class of network traffic, these models are evaluated against the validation set samples to determine their individual F1-scores. A higher F1-score implies lower false positives as well as false negatives and hence is a better evaluation metric compared to the accuracy.

The representative samples' dataset obtained for each class after sample selection is split into $60\%, 20\%, 20\%$ train, validation, and test set [26]. Table 4 shows the performance of individual GMM models trained for each class of the attacks of the CICIDS2017 dataset. The precision (Pr), recall (Re), and F1-score ($F1$) values are obtained on the validation sample set. It can be observed that each trained GMM model can identify the specific class samples with higher F1-scores. These F1-scores are sorted in the descending order to determine the multi-level ordering of classes as follows: **DDoS** $\rightarrow$ **Benign** $\rightarrow$ **Bruteforce** $\rightarrow$ **Bot** $\rightarrow$ **DoS** $\rightarrow$ **WebAttacks** $\rightarrow$ **Portscan**.

For the simplicity of discussion, consider three classes, namely Bruteforce, DoS, and WebAttacks. Their trained GMM models are arranged in this order of decreasing F1-

Table 4: GMM model evaluation per class

Class	H	Pr	Re	$F1$	**Class**	H	Pr	Re	$F1$
Benign	64	0.98	0.98	0.98	Bot	32	1.00	0.91	0.95
BruteForce	64	1.00	0.93	0.96	DDoS	64	1.00	0.98	0.99
DoS	128	1.00	0.88	0.93	Portscan	64	0.61	0.93	0.73
WebAttacks	64	1.00	0.83	0.90					

scores. For each test data sample, first, the Bruteforce GMM model will apply the IQR threshold criteria to determine if the log-likelihood *score* is within its IQR range; if yes, then this sample is labeled as Bruteforce, otherwise, the procedure is repeated for DoS, and WebAttacks GMM trained models. Suppose the log-likelihood score of the test sample does not satisfy the IQR criteria of all three GMM models. In that case, this test sample is denoted as a novel attack or a zero-day attack sample. The proposed multi-level approach can thus determine any novel attacks. This strategy can help security administrators take appropriate action based on the type of attack or novel attacks and take preventive steps.

4.4 Experimental Results and Analysis

Various experiments are conducted to validate the working of the proposed work with respect to binary and multi-class classification and compared with several state-of-the-art existing works detailed as follows.

4.4.1 Binary Classification

For binary classification, the samples of the dataset are classified as either belonging to the Benign class or the Anomalous class that comprises of all attacks. The proposed work is compared against traditional classifiers and various well-known anomaly detection algorithms (LOF, OCSVM, DBN, etc.) from [25], [26], [27]. Table 5 shows the results of binary classification, where it can be observed that the proposed work outperforms the existing techniques for classifying the samples as either Benign or Anomalous.

Table 5: Evaluation result of binary classification

Approach	Pr	Re	$F1$	**Approach**	Pr	Re	$F1$
LOF [25]	0.76	0.89	0.82	OCSVM [25]	0.70	0.80	0.75
IF [25]	0.87	0.75	0.81	RC [25]	0.87	0.72	0.79
DBN [26]	0.90	0.50	0.65	Ensemble [26]	0.89	0.92	0.91
MLP [26]	0.90	0.94	0.92	SVM [27]	0.80	0.70	0.75
Proposed Work	**0.97**	**0.95**	**0.96**				

4.4.2 Multi-class Classification

For the multi-class classification of the test data samples, the proposed multi-level approach is compared with the existing techniques. Table 6(a) shows the results of individual GMM

Table 6: Evaluation results of multi-class classification

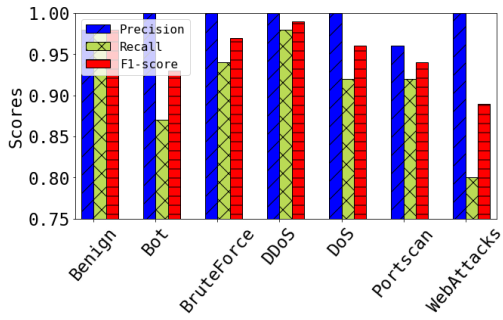
(a) Known attacks classification

Class	Pr	Re	$F1$
Benign	0.98	0.98	0.98
Bot	1.00	0.87	0.93
BruteForce	1.00	0.94	0.97
DDoS	1.00	0.98	0.99
DoS	1.00	0.92	0.96
Portscan	0.96	0.92	0.94
WebAttacks	1.00	0.80	0.89

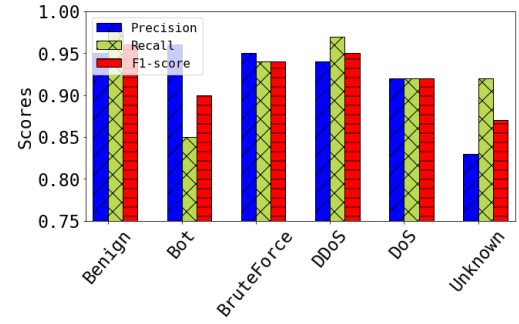
(b) Unknown attacks classification

Class	Pr	Re	$F1$
Benign	0.95	0.98	0.96
Bot	0.96	0.85	0.90
BruteForce	0.95	0.94	0.94
DDoS	0.94	0.97	0.95
DoS	0.92	0.92	0.92
Unknown	0.83	0.92	0.87

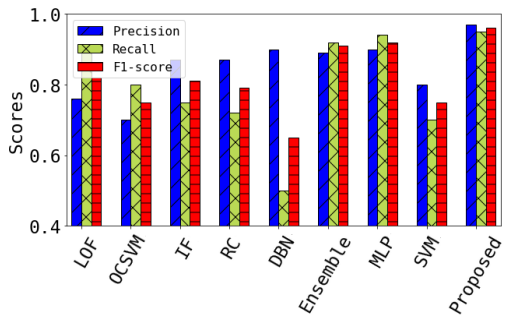
models for the test dataset considering all known network traffic classes, where it can be observed that test samples can be classified effectively with the average F1-score of 0.95.



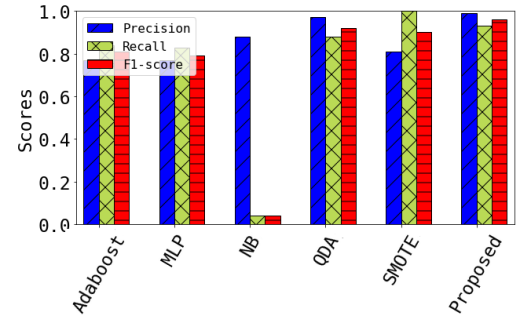
(a) Known attacks classification



(b) Unknown attacks classification



(c) Binary classification



(d) Multi-level classification

Fig. 6: Performance evaluation of proposed work

To determine the performance of the proposed system to detect unknown attacks, the following classes were used in the multi-level classification system: Benign, Bot, BruteForce, DDoS, and DoS. Portscan and WebAttacks are considered to be unknown attacks, i.e., these classes are not considered during multi-level classification. Thus, only five GMM models are used in the testing phase, and for each test data sample, it is assigned the appropriate label if it belongs to the probability distribution of the specific GMM, else it is assigned an Unknown label if it is an outlier to any of the existing GMM models. Table 6(b) illustrates the performance of this scenario where we observe that the proposed

multi-level system can identify the unknown attacks with a high F1-score.

Further, the proposed work is compared with existing multi-class classifiers in the literature and the results are shown in Fig. 6. The results show that the proposed approach achieves the highest F1-score compared to existing approaches.

5 Combating Imbalance Network Traffic Problem with Regularized WGAN

In the case of network traffic, typically, the benign or normal samples frequently occur more than the attack samples; this phenomenon leads to the data imbalance problem in any network traffic dataset. This issue can degrade the classification performance since the conventional approach assigns equal importance to all classes assuming an equal number of samples for each class. However, this assumption fails for imbalanced datasets, and thus the classification performance is adversely affected. In this work, the classification performance of various attacks, including minority attacks, is improved by solving the imbalance problem. This problem is currently solved mainly by using oversampling methods such as synthetic minority ovthe ersampling technique (SMOTE), but the data it generates may not be sufficiently realistic. Instead, a generative adversarial network (GAN) can be used for generating better samples to capture the probability distribution of data. To address mode collapse and unstable training suffered by GAN during the generation of minority attacks, regularized Wasserstein GAN is proposed, and the resulting generated samples are statistically validated. The experiments are conducted on the recent dataset CICIDS 2017 which contains real traces of benign and up-to-date malicious activities of the network traffic.

The architecture of the proposed workflow is shown in Fig. 7, where the imbalanced input data is pre-processed to remove the redundant features and samples, followed by log transformation to reduce the scale of values. The minority instances are augmented using data augmentation methods, and their statistical validity is obtained to determine the most effective augmentation method. These augmented samples are combined with the majority instances resulting in a balanced dataset. Besides the conventional TRTR (Train on Real, Test on Real) classification strategy, the performance is also evaluated using TSTR (Train on Synthetic, Test on Real) and TRTS (Train on Real, Test on Synthetic) strategies to validate the learning capability and diversity.

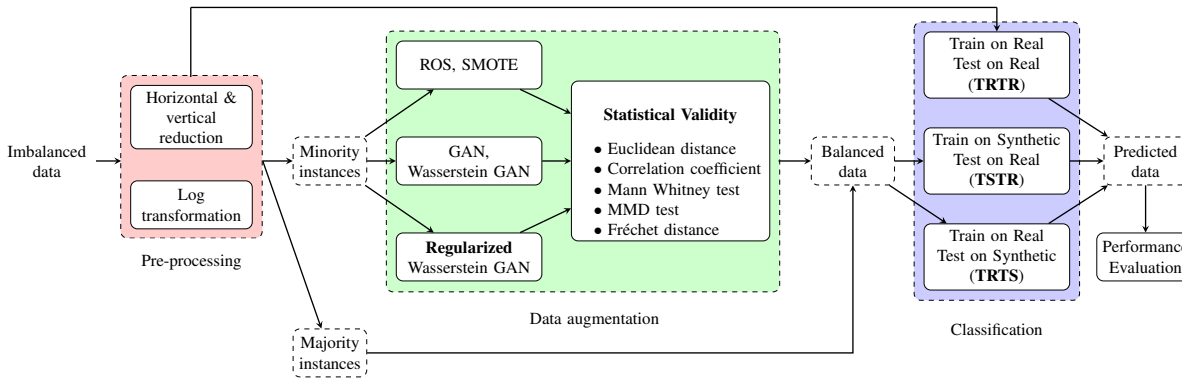


Fig. 7: Architecture of the proposed methodology

5.1 GAN based Data Augmentation

GAN comprises of two competing models, discriminator $\mathcal{C}$ and generator $\mathcal{H}$, in an adversarial setting as illustrated in Fig. 8. From the available set of real imbalanced data, for each attack class, an instance $\mathbf{u}$ is randomly selected and given as input to $\mathcal{C}$ that gives the probability output $\mathcal{C}(\mathbf{u})$ indicating the possibility of $\mathbf{u}$ sampled from the real data probability distribution $\mathbb{P}_t$. A latent vector $\mathbf{q}$ sampled from a known distribution $\mathbb{P}_q$ is fed to $\mathcal{H}$ that generates a synthetic data sample $\mathcal{H}(\mathbf{q})$. This generated sample is given as input to the discriminator for predicting the probability $\mathcal{C}(\mathcal{H}(\mathbf{q}))$ that $\mathcal{H}(\mathbf{q})$ is sampled from the generating data distribution $\mathbb{P}_s$.

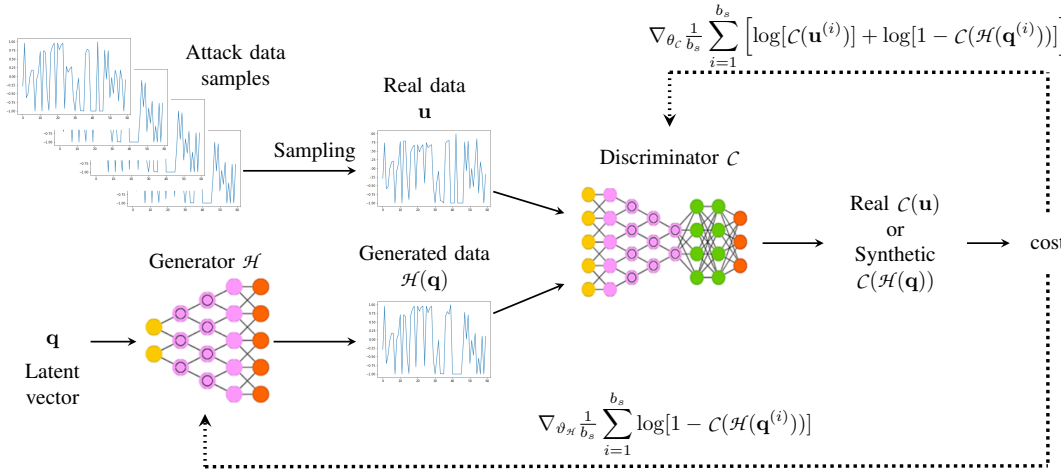


Fig. 8: Architecture of GAN

The objective function of GAN is given by Eq. (7), where the discriminator $\mathcal{C}$ is maximized and the generator $\mathcal{H}$ is minimized, $\mathbb{P}_t$ refers to the probability distribution of the real data, and $\mathbb{P}_q$ refers to the probability distribution of the latent vector (typically, the Gaussian distribution). Accordingly, the loss function of $\mathcal{C}$ is $\mathcal{L}_\mathcal{C} = -\mathbb{E}_{\mathbf{u}\sim\mathbb{P}_t}\left[\log \mathcal{C}(\mathbf{u})\right] - \mathbb{E}_{\mathbf{q}\sim\mathbb{P}_q}\left[\log(1 - \mathcal{C}(\mathcal{H}(\mathbf{q})))\right]$ and that of $\mathcal{H}$ is $\mathcal{L}_\mathcal{H} = \mathbb{E}_{\mathbf{q}\sim\mathbb{P}_q}\left[\log(1 - \mathcal{C}(\mathcal{H}(\mathbf{q})))\right]$.

$$\min_{\mathcal{H}} \max_{\mathcal{C}} \mathbb{E}_{\mathbf{u}\sim\mathbb{P}_t}\left[\log \mathcal{C}(\mathbf{u})\right] + \mathbb{E}_{\mathbf{q}\sim\mathbb{P}_q}\left[\log(1 - \mathcal{C}(\mathcal{H}(\mathbf{q})))\right] \tag{7}$$

The discriminator neural network $\mathcal{C}$ of the GAN model used in this work is shown in Fig. 9 with multiple fully connected hidden dense layers using ReLU (rectified linear unit) activation function and DropOut layer [28] for regularization. The final dense layer uses the sigmoid activation function to determine the probability of the input data sample as real or fake/synthetic. The generator neural network $\mathcal{H}$ is shown in Fig. 10 that takes a r_d dimensional latent vector $\mathbf{q} \in \mathbb{R}^{r_d}$, followed by several dense layers with ReLU activation function and DropOut layer for regularization. The final dense layer uses the tanh activation function to obtain the output generated sample in the range of $[-1, 1]$.

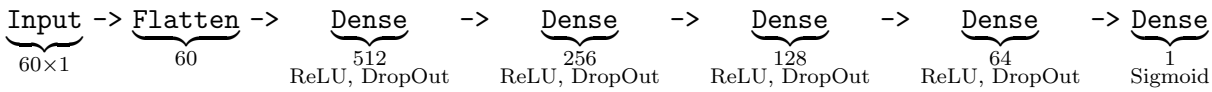


Fig. 9: Discriminator neural network model $\mathcal{C}$ of GAN

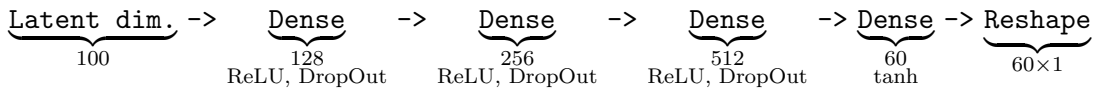


Fig. 10: Generator neural network model $\mathcal{H}$ of GAN

Algorithm 4 shows the pseudocode of the GAN model where lines 1 to 3 implement the loss function of binary cross entropy and the discriminator training is frozen in line 4. The GAN $\mathcal{G}$ model is defined in line 5 using the Keras API [29] notation with the input as the latent vector $\mathbf{q}$ and the output as the discriminator's probability $\mathcal{C}(\mathcal{H}(\mathbf{q}))$. The $\mathcal{G}$ model is compiled using the binary cross-entropy loss function, and Adam optimizer [30] is used for updating the model parameters in line 6.

Algorithm 4 GAN Model

Input: Generator model $\mathcal{H}$, Discriminator model $\mathcal{C}$ $\quad[\mathcal{H}.\text{in} = \mathbf{q}, \mathcal{H}.\text{out} = \mathcal{H}(\mathbf{q})]$

1: **function** BCE($\mathbf{t}_{\text{true}}, \mathbf{t}_{\text{pred}}$) ▷ Binary Cross-Entropy loss function
2: **return** $-\texttt{sum}([\mathbf{t}_{\text{true}}[j] \times \log_2 \mathbf{t}_{\text{pred}}[j]$ **for** j in `range(len(`$\mathbf{t}_{\text{true}}$`))])`
3: $\mathcal{C}$.train $\leftarrow$ False ▷ freeze training of discriminator
4: $\mathcal{G} \leftarrow$ Model($\mathcal{H}$.in, $\mathcal{C}(\mathcal{H}.\text{out})$) ▷ GAN model with $\mathbf{t}_{\text{pred}} = \mathcal{C}(\mathcal{H}.\text{out})$
5: Compile the GAN $\mathcal{G}$ model with BCE loss and the optimizer as ADAM to update the parameters
6: **return** the GAN $\mathcal{G}$ model

Algorithm 5 GAN Training

Input: $\mathcal{H}, \mathcal{C}, \mathcal{G}; \mathbf{U}_{tr}$ (training data), r_d (latent dim.), n_i (# epochs), b_s (batch size)

1: **function** GET_REAL_DATA_SAMPLES($\mathbf{U}_{tr}, b_s$)
2: $\mathbf{U}_r \leftarrow$ the next data batch of size b_s samples from the training data $\mathbf{U}_{tr}$
3: $\mathbf{t}_r \leftarrow$ `ones`$(b_s, 1)$ ▷ assign label +1
4: **return** $\mathbf{U}_r$, $\mathbf{t}_r$ ▷ input to the discriminator $\mathcal{C}$ model
5: **function** GET_LATENT_VECTOR(b_s, r_d)
6: $\mathbf{Q} \leftarrow$ `randn`(b_s, r_d) ▷ draw samples from random normal distribution
7: **return** $\mathbf{Q}$ ▷ input to the generator $\mathcal{H}$ model
8: **function** GET_SYNTHETIC_DATA_SAMPLES ($\mathcal{H}, b_s, r_d$)
9: $\mathbf{Q} \leftarrow$ GET_LATENT_VECTOR(b_s, r_d)
10: $\mathbf{U}_s \leftarrow \mathcal{H}$.`predict`$(\mathbf{Q})$ ▷ $\mathbf{U}_s = \mathcal{H}(\mathbf{Q})$
11: $\mathbf{t}_s \leftarrow$ `zeros`$(b_s, 1)$ ▷ assign label 0
12: **return** $\mathbf{U}_s$, $\mathbf{t}_s$ ▷ input to the discriminator $\mathcal{C}$ model
13: $n_s \leftarrow (\mathbf{U}_{tr}$.`shape`$[0]/b_s) \times n_i$ ▷ number of steps for iteration
14: **for** $i = 1, \ldots, n_s$ **do**
15: $\mathbf{U}_r, \mathbf{t}_r \leftarrow$ GET_REAL_DATA_SAMPLES($\mathbf{U}_{tr}, b_s$)
16: $\mathcal{C}$.`train_on_batch`$(\mathbf{U}_r, \mathbf{t}_r)$ ▷ update the parameters of the discriminator $\mathcal{C}$
17: $\mathbf{U}_s, \mathbf{t}_s \leftarrow$ GET_SYNTHETIC_DATA_SAMPLES($\mathcal{H}, b_s, r_d$)
18: $\mathcal{C}$.`train_on_batch`$(\mathbf{U}_s, \mathbf{t}_s)$ ▷ update the parameters of the discriminator $\mathcal{C}$
19: $\mathbf{Q} \leftarrow$ GET_LATENT_VECTOR(b_s, r_d)
20: $\mathbf{y}_s \leftarrow$ `ones`$(b_s, 1)$ ▷ fool the discriminator $\mathcal{C}$ model by assigning label +1
21: $\mathcal{G}$.`train_on_batch`$(\mathbf{Q}, \mathbf{t}_s)$ ▷ update the parameters of the generator $\mathcal{G}$

Algorithm 5 shows the implementation of training the GAN model for updating the discriminator and generator parameters. A single batch of real samples $\mathbf{U}_r$ is obtained from $\mathbf{U}_{tr}$ with the corresponding output $\mathbf{t}_r$ set to one for all samples in the batch. The synthetic (fake) samples $\mathbf{U}_s$ are generated using the generator $\mathcal{H}$ in lines 8 to 12, where the corresponding output $\mathbf{t}_s$ is set to zero for all synthetic samples in the batch. The

training of GAN happens for n_s iterations, where in each iteration, the real and synthetic samples are used to update the parameters of $\mathcal{C}$.

GAN is known to suffer from the problem of vanishing gradients because of the following characteristics of (discontinuous) KL and JS divergences in the case of disjoint distributions, i.e. $supp(\mathbb{P}_t) \cap supp(\mathbb{P}_s) = \emptyset$: i) if $\mathbb{P}_s(\mathbf{a}) \to 0$ and $\mathbb{P}_t(\mathbf{a}) \to 1$, $KL(\mathbb{P}_s||\mathbb{P}_t) \to 0$, (ii) if $\mathbb{P}_s(\mathbf{a}) \to 1$ and $\mathbb{P}_t(\mathbf{a}) \to 0$, $KL(\mathbb{P}_s||\mathbb{P}_t) \to +\infty$. In both cases, we have $JS(\mathbb{P}_s||\mathbb{P}_t) \to \log 2$, which is a constant resulting in vanishing gradients. To overcome these drawbacks, the objective function of GAN is modified using the Wasserstein distance as explained in the next section.

5.2 WGAN model for data augmentation

The Wasserstein GAN (WGAN) [31] is based on the Wasserstein distance metric (also known as the Earth Mover's distance EMD) defined in Eq. (8), where inf refers to infimum (greatest lower bound), Ψ refers to the joint probability distribution whose marginals are $\mathbb{P}_t$ and $\mathbb{P}_s$. The Wasserstein distance $WD(\mathbb{P}_t, \mathbb{P}_s)$ represents the least amount of effort required to shift the mass from $\mathbb{P}_t$ to $\mathbb{P}_s$, i.e., it measures the discrepancy between these two distributions as is required for the training of GAN model.

$$WD(\mathbb{P}_t, \mathbb{P}_s) = \inf_{\Psi \in \Upsilon(\mathbb{P}_t, \mathbb{P}_s)} \mathop{\mathbb{E}}_{\mathbf{a},\mathbf{b}\sim\Psi} \big[\|\mathbf{a} - \mathbf{b}\|\big] = \sup_{\mathcal{C}} \mathop{\mathbb{E}}_{\mathbf{a}\sim\mathbb{P}_t} [\mathcal{C}(\mathbf{a})] - \mathop{\mathbb{E}}_{\mathbf{b}\sim\mathbb{P}_s} [\mathcal{C}(\mathbf{b})] \tag{8}$$

The dual form is converted to the objective function of WGAN given by Eq. (9) with the constraint that the discriminator, known as the critic for the distance measurement in WGAN, follows 1-Lipschitz continuity. The notation $\|\mathcal{C}\|_L \leq 1$ for the 1-Lipschitz constraint implies that $|\mathcal{C}(\mathbf{a}) - \mathcal{C}(\mathbf{b})| \leq \|\mathbf{a} - \mathbf{b}\| \, \forall \, \mathbf{a}, \mathbf{b}$. In practice, this is achieved by clipping each parameter ϑ of the critic $\mathcal{C}$ to the range $[-c, c]$.

$$\min_{\mathcal{H}} \max_{\substack{\mathcal{C}, \\ \|\mathcal{C}\|_L \leq 1}} \mathop{\mathbb{E}}_{\mathbf{u}\sim\mathbb{P}_t} [\mathcal{C}(\mathbf{u})] - \mathop{\mathbb{E}}_{\mathbf{q}\sim\mathbb{P}_q} [\mathcal{C}(\mathcal{H}(\mathbf{a}))] \tag{9}$$

Algorithm 6 shows the implementation of WGAN, where the $\mathcal{H}$ and $\mathcal{C}$ models for WGAN remain the same as that for GAN with the following differences: (i) the output layer of $\mathcal{C}$ uses a linear activation function; hence it is denoted as a critic model since the output is no longer a probability like in GAN, (ii) the loss function used is Wasserstein loss instead of binary cross-entropy, (iii) the parameters of critic $\mathcal{C}$ are clipped to $[-c, c]$, where typically $c = 0.01$, (iv) the real samples are assigned label -1 and the fake samples are assigned label $+1$, (v) the critic model $\mathcal{C}$ is updated more times than the generator $\mathcal{H}$, and (vi) the RMSProp optimizer is used to achieve stability during the training.

The training implementation for WGAN is similar to that of GAN training with the following differences: (i) the ground truth is set to -1 for the real samples and $+1$ for the fake samples respectively, (ii) the critic is trained n_c times for each training update of the generator, and (iii) the ground truth $\mathbf{t}_s$ for the synthetic samples is set to -1 to fool the critic as the adversarial step.

Algorithm 6 WGAN Model

Input: Generator model $\mathcal{H}$, Critic model $\mathcal{C}$

1: **function** WL($\mathbf{t}_{\text{true}}, \mathbf{t}_{\text{pred}}$) ▷ Wasserstein loss function
2: **return** `mean`([$\mathbf{t}_{\text{true}}[j] \times \mathbf{t}_{\text{pred}}[j]$ **for** j in `range`(`len`($\mathbf{t}_{\text{true}}$))])
3: $\mathcal{C}$.train $\leftarrow$ False ▷ freeze training of critic
4: $\mathcal{WG} \leftarrow$ Model($\mathcal{H}$.in, $\mathcal{C}(\mathcal{H}$.out)) ▷ WGAN model with $\mathbf{t}_{\text{pred}} = \mathcal{C}(\mathcal{H}$.out)
5: Compile the WGAN model $\mathcal{WG}$ with WL loss and the optimizer as RMSProp (and ADAM, for the regularized WGAN $\mathcal{WG}$ models)
6: **return** the WGAN $\mathcal{WG}$ model

5.3 Regularized WGAN Models for Data Augmentation

The parameters of the critic $\mathcal{C}$ are clipped to achieve the 1-Lipschitz constraint in WGAN; however, this leads to limited learning capability since most parameters of the critic model take the value of c as shown in [32]. Also, even though the training process of WGAN is relatively stable and faster than GAN, WGAN has been shown to generate poor-quality samples or fail to converge [32]. Several regularization methods are suggested in the literature to overcome these limitations for modifying the critic loss function. Note that the generator loss function for all regularized WGAN models is unchanged and remains the same as that of WGAN, i.e., $L_{\mathcal{H}}^{\text{Reg. WGAN}} = L_{\mathcal{H}}^{\text{WGAN}}$, since the Lipschitz constraint is applicable only for the critic.

5.3.1 WGAN with Gradient Penalty (WGAN-GP)

The loss function for the critic of the WGAN-GP model [32] is modified as $\mathcal{L}_{\mathcal{C}}^{\mathcal{GP}} = \mathcal{L}_{\mathcal{C}}^{\mathcal{WG}} + \mathcal{W}^{\mathcal{GP}}$, where the regularization term is given by Eq. (10) indicating the gradient penalty to ensure that the critic satisfies the condition of 1-Lipschitz continuity. The gradient is obtained on the linearly interpolated samples $\mathbf{v}$ obtained as $\mathbf{v} = \kappa\mathbf{u} + (1-\kappa)\mathcal{H}(\mathbf{q})$ between the real sample $\mathbf{u}$ and generated sample $\mathcal{H}(\mathbf{q})$, where $\kappa \sim \mathcal{U}(0, 1)$. The gradient penalty (GP) encourages the critic gradient at these sampled points to be closer to one, rather than clipping the parameters to restricted compact space, thus satisfying the 1-Lipschitz constraint. WGAN-GP performs better than WGAN with stable training and faster convergence without the requirement of fine-tuning its hyper-parameters [32].

$$\mathcal{W}^{\mathcal{GP}} = \rho \mathop{\mathbb{E}}_{\mathbf{v}\sim\mathbb{P}_v} \left[\|\nabla_{\mathbf{v}}\mathcal{C}(\mathbf{v})\|_2 - 1\right]^2 \tag{10}$$

5.3.2 Other Existing Regularized WGAN Methods

It is argued in [33] that the interpolated samples $\mathbf{v} \sim \mathbb{P}_{\mathbf{v}}$ do not generally belong to the optimal coupling Υ^*, thus the gradient norms $\|\nabla_{\mathbf{v}}\mathcal{C}(\mathbf{v})\|_2$ may not necessarily match to one, i.e. the regularization effect of WGAN-GP may be too strong. The Lipschitz penalty was thus introduced in [33] to penalize the objective function only if the gradient penalty norm strictly exceeds one, which is a one-sided penalty as compared to WGAN-GP having a two-sided penalty. WGAN-LP is relatively stable during the training with the property of faster convergence compared to WGAN-GP. Also, WGAN-LP is not sensitive to the regularization multiplier as shown in [33]. In [34], the Wasserstein divergence was used to obtain the WGAN-DIV regularization. While WGAN-GP uses the 1-centered gradient

Algorithm 7 Compute_LLP (LOCALIZED LIPSCHITZ PENALTY)

Input: $\mathbf{U}_r, \mathbf{U}_s, b_s, \rho, \mathcal{C}$ model

1: $\boldsymbol{\lambda} \leftarrow \texttt{rand}([b_s, 1])$ ▷ sample numbers for the batch from $\mathcal{U}(0,1)$
2: $\boldsymbol{\xi} \leftarrow \texttt{randn}([b_s, 1])$ ▷ sample numbers for the batch from $\mathcal{N}(\mathbf{0}, l\mathbf{I})$
3: $\mathbf{Z} \leftarrow \boldsymbol{\lambda}\mathbf{U}_r + (1-\boldsymbol{\lambda})(\mathbf{U}_r + \boldsymbol{\xi})$ ▷ obtain linearly interpolated data samples near $\mathbb{P}_t$
4: $\mathbf{E} \leftarrow \texttt{grad}_{\mathbf{Z}}(\mathcal{C}(\mathbf{Z}))$ ▷ compute gradients of the critic model $\mathcal{C}$ with respect to $\mathbf{Z}$
5: $\mathbf{e} \leftarrow \sqrt{\sum_{j=1}^{bs}(\mathbf{E}[j,:])^2}$ ▷ compute the norm of the gradients
6: $llp \leftarrow \texttt{mean}(\texttt{max}(0, (\mathbf{e}-1)^2))$ ▷ compute the localized Lipschitz penalty
7: **return** $\rho \times llp$

penalty term, WGAN-DIV follows the 0-centered penalty term, which has been shown in [35] to improve the learning capacity of the model and also to avoid over-fitting.

A significant drawback of the above regularization methods is that it is generally not feasible to penalize the gradients *everywhere* using a finite number of real and generated samples. Also, the optimal critic model $\mathcal{C}^*$ may not be differentiable everywhere [33]. Thus, these penalties are applicable only on the interpolated samples $\mathbf{v}$ leaving out several parts of the support domain. Notably, during the initial stages of the training, the generated samples $\mathcal{H}(\mathbf{q})$ and hence the interpolated samples $\mathbf{v}$ could be far from the real distribution $\mathbb{P}_t$; thus, the 1-Lipschitz constraint may not be enforced till $\mathbb{P}_s$ becomes closer to $\mathbb{P}_t$. Since the critic is trained using both real and generated samples, it is sufficient to satisfy the 1-Lipschitz constraint closer to the real distribution. Thus, the penalty is not applied on the interpolated samples between real and synthetic samples, but only on the real samples around the data manifold of real probability distribution $\mathbb{P}_t$.

5.3.3 Proposed Regularized WGAN with Localized Lipschitz Penalty (WGAN-LLP)

We propose the WGAN-LLP regularization term by applying the one-sided penalty on the linearly interpolated samples between the real sample $\mathbf{u}$ and the perturbed real sample $\mathbf{u}+\boldsymbol{\xi}$. The regularization term of the WGAN-LLP penalty is given by Eq. (11), where $\mathbf{s} = \lambda\mathbf{u} + (1-\lambda)\mathbf{v}$, $\mathbf{v} = \mathbf{u} + \boldsymbol{\xi}$, $\lambda \sim \mathcal{U}(0,1)$, and $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, l\mathbf{I})$.

$$\mathcal{W}^{\mathcal{LLP}} = \rho \underset{\substack{\mathbf{u}\sim\mathbb{P}_t,\\ \lambda\sim\mathcal{U}(0,1),\\ \boldsymbol{\xi}\sim\mathcal{N}(\mathbf{0},l\mathbf{I})}}{\mathbb{E}} \Big[\max\big\{0, \big\|\nabla_{\mathbf{s}}\mathcal{C}\big[\lambda\mathbf{u} + (1-\lambda)(\mathbf{u}+\boldsymbol{\xi})\big]\big\|_2 - 1\big\} \Big]^2 \tag{11}$$

Algorithm 7 shows the implementation of the proposed LLP regularization term $\mathcal{W}^{\mathcal{LLP}}$. The inputs $\mathbf{U}_r$ and $\mathbf{U}_s$ refer to real and synthetic samples, both of batch size b_s, and ρ is the constant multiplier for the penalty term. In lines 1 and 2, random numbers are sampled from $\mathcal{U}(0,1)$ and $\mathcal{N}(\mathbf{0}, l\mathbf{I})$ for each data sample of the batch, and linearly interpolated samples are obtained in line 3. The gradient of the critic scores with respect to the linearly interpolated samples is obtained in line 4, and the gradient norm for each sample in the batch is obtained in line 5. Line 6 computes the average penalty for all samples in the batch as per Eq. (10).

Algorithm 8 shows the implementation of WGAN-LLP training using the generator $\mathcal{H}$ and critic $\mathcal{C}$ models and the input training data. The critic loss $\mathcal{L}_{\mathcal{C}}^{\mathcal{WG}}$ is obtained in lines 1

Algorithm 8 WGAN-LLP Training

Input: models $\mathcal{H}, \mathcal{C}, \mathbf{U}_{tr}$ (training data), r_d, b_s, n_c, n_i, ρ

1: **function** CRITIC_LOSS($\mathbf{s}_{\text{out}}, \mathbf{t}_{\text{out}}$)
2: $t_{\text{loss}} \leftarrow \texttt{mean}(\mathbf{t}_{\text{out}})$ ▷ loss due to real data
3: $s_{\text{loss}} \leftarrow \texttt{mean}(\mathbf{s}_{\text{out}})$ ▷ loss due to fake data
4: **return** $s_{\text{loss}} - t_{\text{loss}}$
5: **function** GENERATOR_LOSS($\mathbf{s}_{\text{out}}$)
6: $s_{\text{loss}} \leftarrow \texttt{mean}(\mathbf{s}_{\text{out}})$ ▷ loss due to fake data
7: **return** $-s_{\text{loss}}$
8: $n_s \leftarrow (\mathbf{U}_{tr}.\texttt{shape}[0]/b_s) \times n_i$ ▷ # iteration steps
9: **for** $m = 1, \ldots, n_s$ **do**
10: **for** $n = 1, \ldots, n_c$ **do**
11: $\mathbf{U}_r \leftarrow$ GENERATE_REAL_SAMPLES($\mathbf{U}_{tr}, b_s$)
12: $\mathbf{U}_s \leftarrow$ GENERATE_FAKE_SAMPLES($\mathcal{H}, b_s, r_d$)
13: $\mathbf{s}_{\text{out}} \leftarrow \mathcal{C}(\mathbf{U}_s)$ ▷ output of $\mathcal{C}$ for fake data
14: $\mathbf{t}_{\text{out}} \leftarrow \mathcal{C}(\mathbf{U}_r)$ ▷ output of $\mathcal{D}$ for real data
15: $cost_c \leftarrow$ CRITIC_LOSS($\mathbf{s}_{out}, \mathbf{t}_{out}$)
16: $llp \leftarrow$ COMPUTE_LLP($\mathbf{U}_r, \mathbf{U}_s, b_s, \rho, \mathcal{C}$)
17: $loss_c \leftarrow cost_c + llp$ ▷ total loss of $\mathcal{C}$
18: Apply gradients $\nabla_{loss_c}(\mathcal{C})$ to update $\mathcal{C}$
19: $\mathbf{U}_s \leftarrow$ GENERATE_FAKE_SAMPLES($\mathcal{H}, b_s, r_d$)
20: $\mathbf{s}_{\text{out}} \leftarrow \mathcal{C}(\mathbf{U}_s)$ ▷ output of $\mathcal{C}$ for fake data
21: $loss_h \leftarrow$ GENERATOR_LOSS($\mathbf{s}_{\text{out}}$) ▷ loss of $\mathcal{H}$
22: Apply gradients $\nabla_{loss_h}(\mathcal{H})$ to update $\mathcal{H}$

to 4 and the generator loss $\mathcal{L}_{\mathcal{H}}^{\mathcal{WG}}$ is obtained in lines 5 to 7. For each iteration step m of the training from lines 9 to 22, the critic is trained multiple times (n_c) for each generator update. The LLP penalty term computed in line 16 is added to the critic loss to determine the overall regularized critic loss, using which the critic parameters are updated in line 18 with the Adam optimizer. The generator parameters are updated using the Adam optimizer in line 22 with the generator loss obtained in line 21.

An illustration of the existing and proposed regularization techniques for WGAN is shown in Fig. 11, where the real $\mathbb{P}_t$ and generating $\mathbb{P}_s$ probability distributions are shown along with the support regions for obtaining the linearly interpolated samples. For WGAN-GP, WGAN-LP, and WGAN-DIV, the support region is in between $\mathbb{P}_t$ and $\mathbb{P}_s$, whereas for WGAN-DR and WGAN-LLP, the support region is close to $\mathbb{P}_t$. For the proposed WGAN-LLP, the penalty is computed on the interpolated samples that are more closer to $\mathbb{P}_t$ relative to WGAN-DR.

5.4 Statistical Validity of Data Augmentation Models

We verify the statistical validity of the generated samples using the above-mentioned data augmentation methods with various statistical measures. Consider the real data samples $\mathbf{P} = \{\mathbf{p}_i\}_{i=1}^{N_P} \sim \mathbb{P}_t$ and the generated data samples $\mathbf{Q} = \{\mathbf{q}_j\}_{j=1}^{N_Q} \sim \mathbb{P}_s$, where $\mathbf{p}_i, \mathbf{q}_j \in \mathbb{R}^F$. The average of the real and generated data samples is computed as $\boldsymbol{\nu}_{\mathbf{P}} = \frac{1}{N_P} \sum_{i=1}^{N_P} \mathbf{p}_i$ and

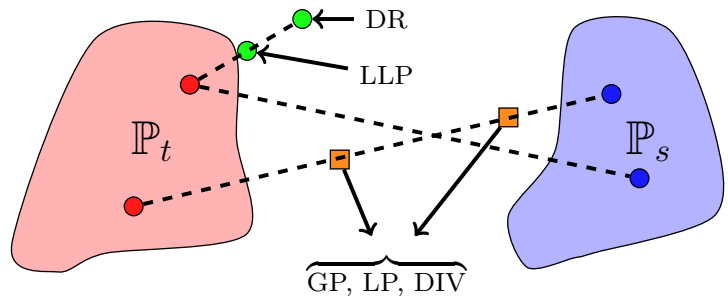


Fig. 11: Illustration of regularization methods

$\boldsymbol{\nu}_{\mathbf{Q}} = \frac{1}{N_Q}\sum_{j=1}^{N_Q}\mathbf{q}_j$, respectively.

a) Euclidean distance: The Euclidean distance is measured between the average real and average generated data as $ED(\mathbf{P},\mathbf{Q}) = \|\boldsymbol{\nu}_{\mathbf{P}} - \boldsymbol{\nu}_{\mathbf{Q}}\|^2 = \sum_{f=1}^{F}(\nu_{\mathbf{P},f} - \nu_{\mathbf{Q},f})^2$. Lower ED indicates that the real and generated samples are similar.

b) Correlation Coefficient: The Pearson correlation coefficient (CC) measures the similarity between the average real and average generated data as $CC(\mathbf{P},\mathbf{Q}) = \frac{F\sum_f \nu_{\mathbf{P},f}\nu_{\mathbf{Q},f} - \sum_f \nu_{\mathbf{P},f}\sum_f \nu_{\mathbf{Q},f}}{\sqrt{F\sum_f \nu_{\mathbf{P},f}^2 - (\sum_f \nu_{\mathbf{P},f})^2} \times \sqrt{F\sum_f \nu_{\mathbf{Q},f}^2 - (\sum_f \nu_{\mathbf{Q},f})^2}}$. Since $-1 \leq CC \leq 1$, a higher value of CC indicates that the real and generated samples are similar.

c) Mann Whitney Test: The Mann-Whitney test determines the U statistic given by $U(\mathbf{P},\mathbf{Q}) = \sum_{f=1}^{F}\sum_{f'=1}^{F}\texttt{bool}(\nu_{\mathbf{P},f} < \nu_{\mathbf{Q},f'})$. If $\mathbb{P}_t$ and $\mathbb{P}_s$ represents similar distributions, then the expected value of U is $\frac{F^2}{2}$, since the values below some rank will occur as often for $\nu_{\mathbf{P}}$ as for $\nu_{\mathbf{Q}}$. The alternate hypothesis $H_1 : \mathbb{P}_t \neq \mathbb{P}_s$ is rejected if the resulting p-value of the U statistic is less than 5% significance level.

d) Maximum Mean Discrepancy: The maximum mean discrepancy (MMD) test [36], given by Eq. (12), determines if the real samples from $\mathbb{P}_t$ and the generated samples from $\mathbb{P}_s$ are similar by using the kernel $\mathbf{Z} : \mathbf{P} \times \mathbf{Q} \to \mathbb{R}$. Typically, Gaussian kernel is used given by $Z(\mathbf{p}_i,\mathbf{q}_j) = \exp(-\frac{1}{2\omega^2}\|\mathbf{p}_i - \mathbf{q}_j\|^2)$, where ω^2 is obtained as the median of the pairwise distances between $\mathbf{P}$ and $\mathbf{Q}$.

$$\begin{aligned} MMD^2(\mathbf{P},\mathbf{Q}) &= \|\boldsymbol{\nu}_{\mathbf{P}} - \boldsymbol{\nu}_{\mathbf{Q}}\|_{\mathcal{H}}^2 = \langle\boldsymbol{\nu}_{\mathbf{P}},\boldsymbol{\nu}_{\mathbf{P}}\rangle + \langle\boldsymbol{\nu}_{\mathbf{Q}},\boldsymbol{\nu}_{\mathbf{Q}}\rangle - 2\langle\boldsymbol{\nu}_{\mathbf{P}},\boldsymbol{\nu}_{\mathbf{Q}}\rangle \\ &= \overline{\mathbf{Z}}_{\mathbb{P}_t,\mathbb{P}_t} + \overline{\mathbf{Z}}_{\mathbb{P}_s,\mathbb{P}_s} - 2\overline{\mathbf{Z}}_{\mathbb{P}_t,\mathbb{P}_s} \end{aligned} \tag{12}$$

For the null hypothesis $H_0 : \mathbb{P}_t \approx \mathbb{P}_s$ and the alternate hypothesis $H_1 : \mathbb{P}_t \neq \mathbb{P}_s$, if the empirical MMD is far from zero (i.e. the distributions are far apart), then reject H_1, otherwise accept H_0.

e) Fréchet Distance: The Fréchet distance (FD) is measured as given by Eq. (13), where $\boldsymbol{\Gamma}_{\mathbf{P}}$ and $\boldsymbol{\Gamma}_{\mathbf{Q}}$ are the covariance matrices of $\mathbf{P}$ and $\mathbf{Q}$, respectively, and $\texttt{Tr}$ is the trace operator. The Fréchet distance also considers the shape of the data distributions using the covariance matrices and is thus a robust measure relative to the Euclidean distance. Lower FD indicates that the real and generated samples are similar.

$$FD(\mathbf{P},\mathbf{Q}) = \|\boldsymbol{\nu}_{\mathbf{P}} - \boldsymbol{\nu}_{\mathbf{Q}}\|^2 + \texttt{Tr}\left(\boldsymbol{\Gamma}_{\mathbf{P}} + \boldsymbol{\Gamma}_{\mathbf{Q}} - 2\sqrt{\boldsymbol{\Gamma}_{\mathbf{P}}\boldsymbol{\Gamma}_{\mathbf{Q}}}\right)] \tag{13}$$

5.5 Classification Strategies

For the evaluation of binary and multi-class classification for imbalanced as well as balanced datasets, three classification strategies are proposed: TRTR (Train on Real, Test on Real), TSTR (Train on Synthetic, Test on Real), and TRTS (Train on Real, Test on Synthetic). While TRTR is used to classify the original imbalanced dataset, TSTR and TRTS are used for performance evaluation on the balanced dataset, obtained by augmenting with the generated samples. TRTR does not use synthetic data for evaluation; however, it serves as a baseline for performance comparison between the imbalanced and balanced (augmented) datasets. A higher TSTR performance implies that the augmentation method avoids the problem of mode collapse and can generate diverse samples. A higher TRTS performance implies that the augmentation method can generate realistic-looking data samples. The scores are obtained using appropriate evaluation metrics; namely, precision Pr, recall Re and F1-score $F1$, given by Eq. (14). Precision denotes the ratio of correctly predicted samples to total positively predicted samples, recall denotes the ratio of correctly predicted samples to actual positive samples in the dataset, and the F1 score is the harmonic mean of precision and recall.

$$Pr = \frac{TP}{TP + FP}, \; Re = \frac{TP}{TP + FN}, \; F1 = \frac{2PR}{P + R} \tag{14}$$

5.6 Experimental Results and Analysis

The following pre-processing steps are performed on the CICIDS2017 dataset: i) horizontal and vertical reduction, and ii) feature transformation. The quantitative results of all nine data augmentation methods (ROS, SMOTE, GAN, WGAN, WGAN-GP, WGAN-LP, WGAN-DIV, WGAN-DR, and WGAN-LLP) are shown in the paper (skipped here for brevity) for each attack class of the CICIDS2017 dataset with respect to five statistical validity measures, i.e., ED, CC, U, MMD, and FD. A sample result is shown in Tables 7 to 10 for Bot, Slowloris, Slowhttp, and WebAttacks traffic attacks. For each attack class, WGAN-LLP achieves the highest CC and lowest ED, MMD, and FD scores, and thus WGAN-LLP shows statistically significant improvement for all attack classes.

Table 7: Statistical evaluation of data augmentation methods for Bot attack

Method	ED	CC	U (p-value)	MMD	FD
ROS, SMOTE	0.45, 0.48	0.95, 0.93	1671 (0.5), 1787 (0.9)	0.04, 0.05	1.44, 1.58
GAN, WGAN	1.37, 0.15	0.79, 0.99	1394 (0.03), 1766 (0.86)	0.47, 0.05	4.68, 0.89
WGAN-GP, LP	0.23, 0.16	0.98, 0.99	1797 (0.99), 1773 (0.89)	0.02, 0.01	0.16, 0.12
WGAN-DIV, DR	0.12, 0.11	0.99, 0.99	1789 (0.96), 1749 (0.79)	0.01, 0.01	0.08, 0.08
WGAN-LLP	**0.07**	**0.99**	1742 (0.76)	**0.003**	**0.03**

A subset of the qualitative comparison of the real and generated samples is illustrated in Fig. 12, where for each attack class, the average of original samples and the samples generated by GAN and WGAN-LLP are plotted. For all attack classes, WGAN-LLP results in the average generated sample closer to the original average with minimal loss, and thus for further experiments, the samples generated by WGAN-LLP are used to obtain the balanced dataset.

Table 8: Statistical evaluation of data augmentation methods for Slowloris attack

Method	ED	CC	U (p-value)	MMD	FD
ROS, SMOTE	0.39, 0.37	0.98, 0.98	1696 (0.58), 1718 (0.67)	0.11, 0.10	0.67, 0.51
GAN, WGAN	1.88, 0.34	0.83, 0.98	1496 (0.11), 1778 (0.91)	0.13, 0.11	7.55, 1.45
WGAN-GP, LP	0.35, 0.36	0.98, 0.98	1727 (0.70), 1780 (0.91)	0.024, 0.01	0.92, 0.82
WGAN-DIV, DR	0.39, 0.22	0.98, 0.99	1656 (0.45), 1692 (0.57)	0.01, 0.02	0.67, 0.22
WGAN-LLP	**0.14**	**0.99**	1760 (0.83)	**0.008**	**0.21**

Table 9: Statistical evaluation of data augmentation methods for Slowhttp attack

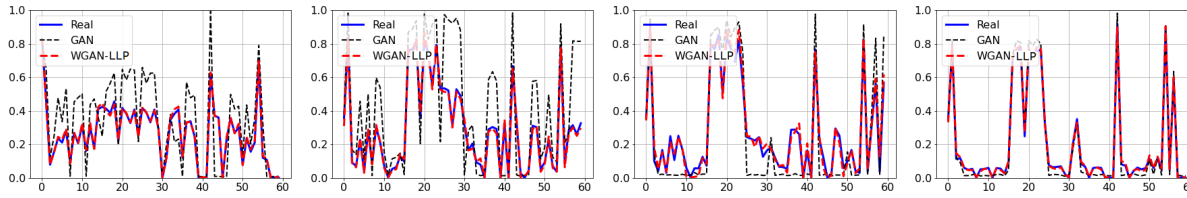
Method	ED	CC	U (p-value)	MMD	FD
ROS, SMOTE	0.45, 0.51	0.98, 0.97	1688 (0.55), 1702 (0.60)	0.02, 0.03	0.79, 0.70
GAN, WGAN	1.12, 0.35	0.93, 0.98	1161 (0.001), 1682 (0.53)	0.42, 0.07	5.01, 1.45
WGAN-GP, LP	0.35, 0.38	0.99, 0.98	1701 (0.60), 1797 (0.99)	0.02, 0.02	1.21, 1.24
WGAN-DIV, DR	0.31, 0.26	0.99, **0.99**	1712 (0.64), 1788 (0.95)	0.02, 0.01	0.95, 0.79
WGAN-LLP	**0.26**	0.99	1789 (0.95)	**0.01**	**0.66**

Table 10: Statistical evaluation of data augmentation methods for Webattacks

Method	ED	CC	U (p-value)	MMD	FD
ROS, SMOTE	0.44, 0.48	0.95, 0.93	1671 (0.50), 1787 (0.95)	0.04, 0.05	1.43, 1.57
GAN, WGAN	1.37, 0.15	0.78, 0.99	1394 (0.03), 1766 (0.86)	0.47, 0.05	4.68, 0.89
WGAN-GP, LP	0.22, 0.16	0.98, 0.99	1797 (0.99), 1773 (0.88)	0.01, 0.01	0.16, 0.11
WGAN-DIV, DR	0.12, 0.10	0.99, 0.99	1789 (0.95), 1749 (0.79)	0.006, 0.006	0.08, 0.08
WGAN-LLP	**0.07**	**0.99**	1742 (0.76)	**0.003**	**0.03**

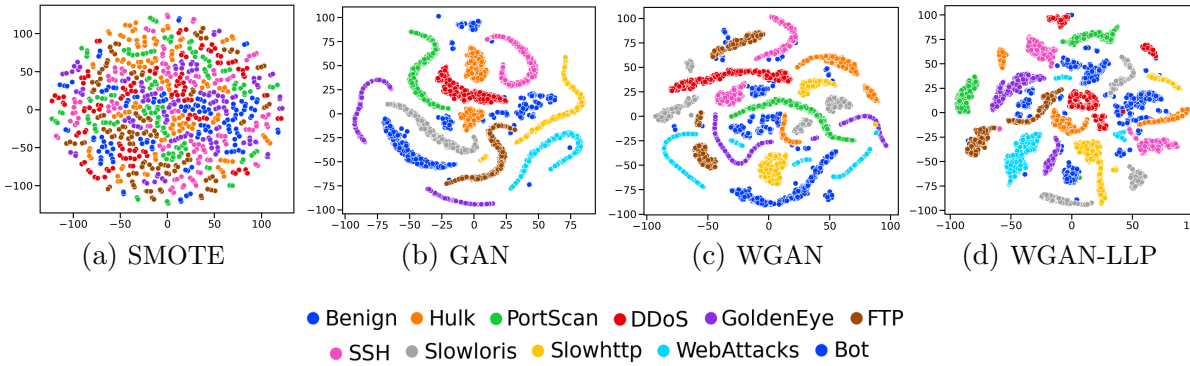
The t-SNE (t-distributed stochastic neighbor embedding) [37] visualization of the generated samples is illustrated in Fig. 13 for SMOTE, GAN, WGAN, and WGAN-LLP augmentation methods. Note that for the Benign class, the original samples are used for t-SNE visualization. In the projected two-dimensional space by t-SNE, SMOTE performs worst. In contrast, WGAN-LLP results in a better separation of attack classes, which is helpful for better discrimination capability of the classifiers for multi-class classification.

The support vector machine (SVM) classifier is used in this work for binary and multi-class classification; its hyper-parameters (cost penalty and kernel parameters) are obtained using grid-search on the training data for each experiment. While GAN performs better than ROS and SMOTE, its performance improves by using the Wasserstein loss (WGAN). The performance of WGAN is further improved by using regularization methods; specifically, WGAN-LLP performs better than other regularizers. The best F1-score of 0.98 is obtained by the WGAN-IDR method in the balanced dataset with $BR = 1:1$ as illustrated in Fig. 14; it is observed that WGAN-LLP outperforms other augmentation methods. Table 11 shows the classification report for binary classification using TRTR, TRTS, and TSTR strategies. The TRTR strategy has the lowest performance due to an imbalanced dataset, which is improved by using the balanced dataset obtained with the WGAN-LLP augmentation method. Both TSTR and TRTS strategies show an improvement in performance, implying that the generated attack samples are diverse



(a) Bot (b) Slowloris (c) Slowhhttp (d) Webattacks

Fig. 12: Comparison of real v/s generated samples



(a) SMOTE (b) GAN (c) WGAN (d) WGAN-LLP

Fig. 13: t-SNE visualization of generated samples

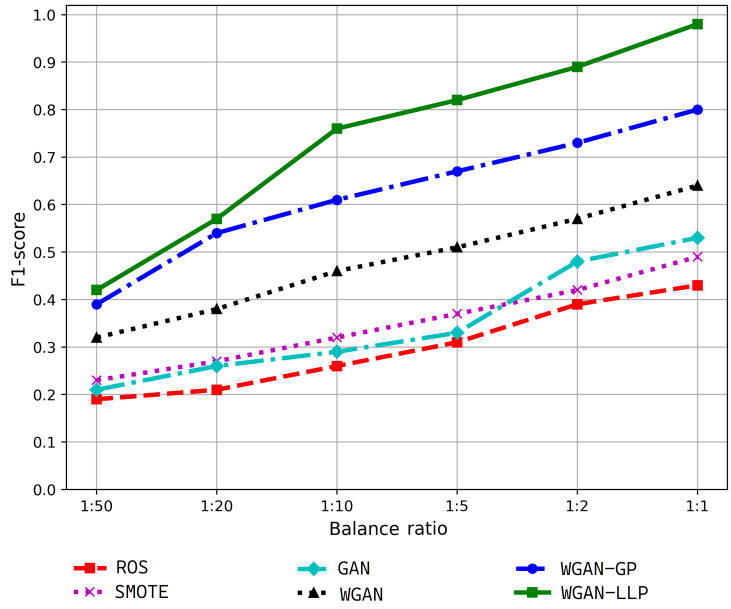


Fig. 14: Performance across various balance ratios

and realistic. Further, the binary classification of the proposed work is compared against existing techniques used to solve the imbalance problem. It can be observed that the proposed work of solving the imbalance problem using the regularized WGAN outperforms the existing techniques for classifying malicious and non-malicious network traffic.

For multi-class classification, the samples are identified as belonging to the Benign class or specific attack classes. Table 12 shows the classification report for multiclass classification using TRTR and TRTS strategies (results of TSTR are shown in the paper).

Table 11: Binary classification performance for CICIDS dataset

Method	**Category**	*Pr*	*Re*	*F*1
Majority	Benign	1.00	0.99	0.99
TRTR	Attack	0.88	0.38	0.53
TSTR	Attack	1.00	0.99	0.99
TRTS	Attack	0.98	0.97	0.97

Method	*Pr*	*Re*	*F*1
SVM [38]	0.80	0.70	0.65
One-class SVM [39]	0.70	0.80	0.75
LR [17]	0.68	0.85	0.75
RF [17]	0.84	0.96	0.90
Adaboost [40]	0.81	0.95	0.88
Proposed	**1.00**	**0.99**	**0.99**

Table 12: Multi-class classification performance for CICIDS dataset

Method	**Category**	*Pr*	*Re*	*F*1	**Method**	**Category**	*Pr*	*Re*	*F*1
TRTR Imbalanced Data	Hulk	0.98	0.96	0.97	TSTR Balanced Data	Hulk	0.97	0.98	0.98
	PortScan	0.99	1.00	0.99		PortScan	0.99	0.99	0.99
	DDoS	1.00	1.00	1.00		DDoS	0.99	1.00	0.99
	GoldenEye	1.00	0.71	0.83		GoldenEye	1.00	0.98	0.99
	FTP	0.95	0.75	0.84		FTP	0.99	0.99	0.99
	SSH	1.00	0.17	0.29		SSH	0.95	0.97	0.96
	Slowloris	1.00	0.76	0.87		Slowloris	0.97	0.99	0.98
	Slowhttp	0.71	0.75	0.73		Slowhttp	0.98	0.97	0.97
	WebAttacks	0.11	0.07	0.09		WebAttacks	0.98	0.98	0.98
	Bot	0.05	0.08	0.06		Bot	0.97	0.98	0.97

Like binary classification, the TRTR strategy has the lowest performance due to imbalance among various attack classes, especially, Bot, WebAttacks, Slowhttp, Slowloris, SSH, FTP, and GoldenEye attacks (highlighted in dashed red). Both TSTR and TRTS strategies improve upon TRTR with increased precision, recall, and F1 score for all minority attack classes.

Table 13: Comparison of multi-class classification with existing work

Method	*Pr*	*Re*	*F*1	**Method**	*Pr*	*Re*	*F*1
AdaBoost [41]	0.77	0.84	0.81	MLP [41]	0.77	0.83	0.79
NB [41]	0.88	0.04	0.04	QDA [41]	0.97	0.88	0.92
BMCD + MLP [8]	-	-	0.81	BMCD + NB [8]	-	-	0.90
SMOTE + LDA [4]	0.91	0.91	0.91	SMOTE + RF [42]	0.88	0.87	0.87
Proposed	**0.99**	**0.97**	**0.98**				

Table 13 shows the performance of existing and proposed work for classification of the various attacks of the CICIDS2017 dataset. It can be observed that the proposed work results in the highest precision, recall, and F1 scores relative to the existing work.

6 Malicious Traffic Detection using Convolutional Neural Network

The major challenges for malicious network activity detection are a huge volume of network traffic, diversity due to new attacks, and reduced performance of low-frequency attacks due to a high imbalance between various classes of attacks. Deep learning techniques can help to tackle these challenges as they have the ability to model complex relationships and concepts using multiple levels of representation. The research question that we address in this section is: Can deep learning models improve the accuracy to detect malicious network traffic and enhance the performance of the system? We have experimented with standard machine learning classifiers as well as deep learning architectures and found a significant performance improvement with deep one-dimensional convolutional neural network (CNN) architecture. The experiments were performed on the benchmark UNSW-NB15 dataset.

6.1 Pre-processing of UNSW-NB15 Dataset

The following pre-processing steps are applied on the dataset:

a) *Symbolic data pre-processing*: The UNSW-NB15 dataset contains $258,726$ data samples with numeric as well as 3 symbolic features: protocol, service, and state. One-hot encoding is used to map these symbolic features into numeric data resulting in feature vector $\mathbf{v} \in \mathbb{R}^{175}$.
b) *Data cleaning*: Since the service feature has many missing values, we dropped this feature to avoid irrelevant information. The `is_ftp_login` being a Boolean feature can have values either 0 or 1. However, there are few samples in the dataset whose `is_ftp_login` feature value is 4 or 2. We have dropped these erroneous samples from the dataset.
c) *Data normalization*: The dataset contains values that varies in a wide range for several features; to deal with this issue, normalization is performed.

6.2 Fully-connected Feed-forward Network

Fig. 15 shows a typical fully-connected feed-forward neural network (FFNN) that comprises of one input layer to which the network features are fed, one or more hidden layers and one output layer with class probabilities indicating whether the data is normal or malicious as well as the category of attack. Hidden layers can automatically extract non-linear features from the data due to the use of non-linear activation functions.

The following network topologies are used in this work:

i) DNN_1: Dense neural network with one hidden layer consisting of 100 neurons,
ii) DNN_2: Dense neural network with two hidden layers consisting of 100 neurons and 50 neurons, and
iii) DNN_3: Dense neural network with three hidden layers consisting of 100 neurons, 50 neurons, and 20 neurons.

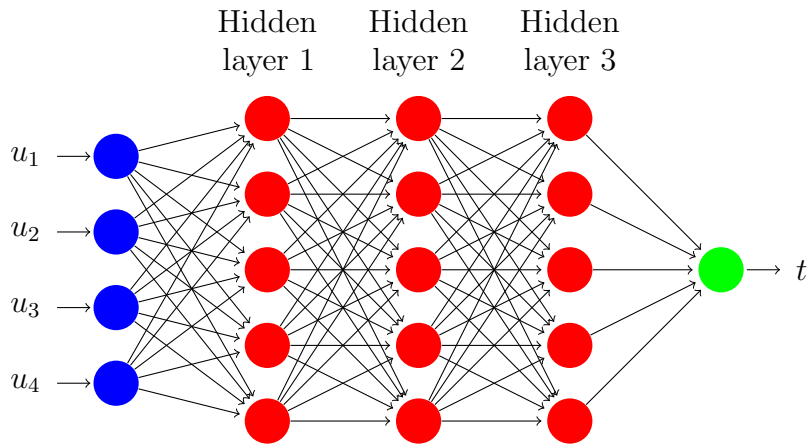


Fig. 15: Illustration of a typical dense neural network architecture

Input and hidden layers: The input layer consists of 175 neurons since we have 175 features per data sample. The neurons in the input layer to the hidden layer and hidden to output layer are fully connected. The hidden layers use ReLU as the non-linear activation function given by $\sigma(h_i) = \max(0, h_i)$ for some output $\mathbf{h}$ of the hidden layer. The back-propagation mechanism is used to train the FFNN architecture using stochastic gradient descent (SGD) optimization variant. The loss function utilized is categorical cross entropy given by $\mathcal{L}_{CCE} = -\sum_{l=1}^{L} t_{m,l} \log q_{m,l}$, where $L = 10$ is the number of classes, $t_{m,l}$ is the indicator variable of the m^{th} sample for the l^{th} class and $q_{m,l}$ denotes the predicted probability.

Regularization: To avoid over-fitting, the dropout regularization method is used where certain neurons are randomly unplugged making the model robust [28]. We have set the dropout probability value to 0.25, i.e. 25% of the neurons are randomly dropped.

Output layer and classification: The output layer consists of 10 neurons, 1 for Normal and 9 for various attack categories. The softmax activation function given by $\sigma(h_i) = \frac{\exp(h_i)}{\sum_{i'} \exp(h_{i'})}$ is used to obtain the class probabilities at the output layer.

6.3 One-dimensional Convolutional Neural Network

Convolutional neural network (CNN) consists of a sequence of two operations, convolution and pooling (sub-sampling), applied repeatedly on the input data. The abstract representation of the CNN architecture is `Input -> Conv_1D -> MaxPool -> Conv_1D -> MaxPool -> Dense -> Output`, where `Conv_1D` represents a one-dimensional convolutional layer, `MaxPool` is the max-pooling layer, and `Dense` refers to a fully-connected hidden layer. The CNN architecture is similar to DNN except the `Conv_1D` and `MaxPool` layers. We have specifically chosen 1D convolution since the input data for the problem of detection of malicious traffic from the network is one dimensional.

Let $\mathbf{u}_m \in \mathbb{R}^F$ be the normalized feature vector of $F = 175$ dimensions for the m^{th} data sample, and $t_m \in \{1, \ldots, L\}$ be the class labels with $L = 10$ classes. The convolutional layer consists of various filters, where each filter learns a specific pattern from the data producing feature maps. Let b be the filter of size n and define $n' = (n-1)/2$. 'Same' padding is used to retain the output shape same as that of the input shape (i.e. zeros

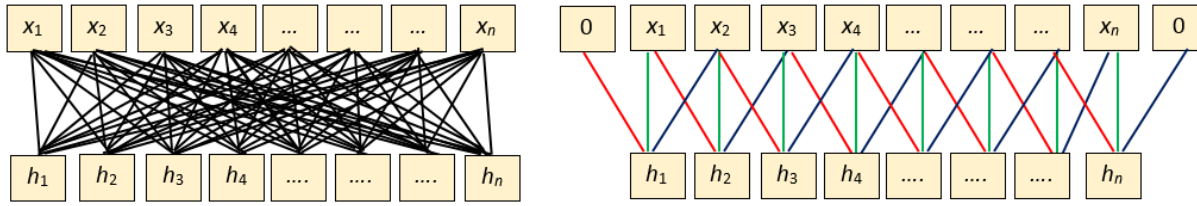


Fig. 16: Connections of FFNN (left) and 1D CNN (right)

are padded to the input). For the c^{th} layer, the output $\mathbf{g}_c$ of the convolution operation is given by Eq. (15), where r_c is the bias. The convolutional layer significantly reduces the number of trainable parameters relative to the fully-connected layers of DNN architecture, as illustrated in Fig. 16, and the filter weights are shared between connections. The max-pooling layer further reduces the complexity by down-sampling the output of the convolutional layer.

$$g_c(k) = \sum_{i=-n'}^{n'} b_c(k) \times u(k+i) + r_c \tag{15}$$

In this work, for detecting the malicious traffic categories of the UNSW-NB15 dataset, we propose two CNN architectures:

- a shallow CNN (S_CNN) comprising of only one `Conv_1D` layer and corresponding one `MaxPool` layer as illustrated in Fig. 17, and
- a deep CNN (D_CNN) comprising of several `Conv_1D` and corresponding `MaxPool` layers as shown in Fig. 18.

For both S_CNN and D_CNN neural network architectures, we reshape the input traffic data so that it can be used as a tensor for the convolution layer. Thus, the feature data matrix $\mathbf{U} \in \mathbb{R}^{M \times F}$ is reshaped to obtain a tensor $\mathbf{U} \in \mathbb{R}^{M \times F \times 1}$. Here, the number of data samples is M and the dimensionality of features is F.

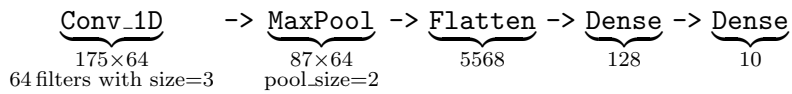


Fig. 17: Proposed shallow 1D CNN (S_CNN) architecture

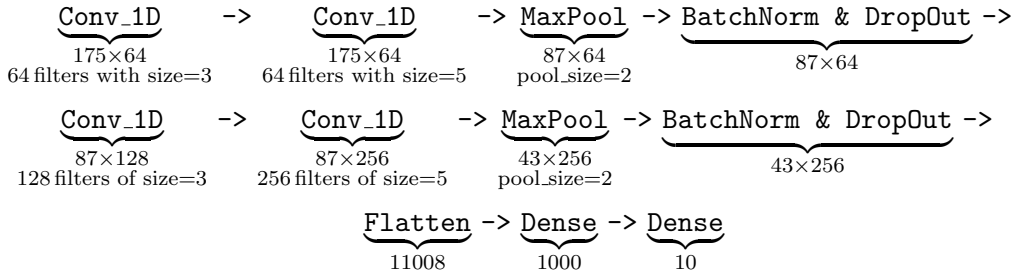


Fig. 18: Proposed deep 1D CNN (D_CNN) architecture

For the `Conv_1D` (feature detector) layer, only defining one filter would allow the neural network to learn only one single feature. This may not be sufficient, thus we define various

number of filters with varying filter size to learn multiple patterns occurring in the data. The `MaxPool` layer downsamples the output of the convolutional layer by 2, i.e. for every 2 samples of the output, the maximum value is retained. Further, batch normalization (`BatchNorm`) and dropout operations are applied in the D-CNN architecture to speed up learning and to avoid over-fitting, respectively. All layers use the ReLU activation function except for the last layer. For the final `Dense` layer, the softmax activation function is used to compute the probability of the predicted classes. The network parameters are learned using back-propagation with the Adam optimization [30].

6.4 Experimental Results and Analysis

The performance of malicious network traffic classification of the UNSW-NB15 dataset is evaluated using standard machine learning (ML) classifiers as well as the proposed neural network architectures. The following standard ML classifiers are evaluated: Logistic Regression (LR), K-Nearest Neighbors (KNN), Naive Bayes (NB), SVM with linear kernel (SVML), SVM with radial basis function kernel (SVMR), Decision Trees (DT), and Random Forest (RF). These are compared with the following neural network architectures: DNN_1, DNN_2, DNN_3, S_CNN, and D_CNN, as well as the existing 2D CNN method of [18]. To avoid the imbalanced class problem, we use a random subset M_l of the UNSW-NB15 dataset for training as given by Eq. (16), where T_l is the number of training samples for class l. The testing subset is also similarly randomly chosen from the original test samples.

$$M_l = \min(2000, \max(T_l)), \quad \text{for } l = 0, \ldots, 9 \tag{16}$$

For the standard ML classifiers, the default parameters were used with the Scikit-learn API. For the neural network architectures, a batch size of 256 was used and the training was executed for 200 epochs using Keras API with Tensorflow backend. The F1-score of the standard ML classifiers is shown in Table 14, where we observe that Logistic Regression (LR) performs better than the other classifiers for most attacks. Analysis, Backdoor, DoS, Shellcode and Worms are the most challenging attack categories to classify and most of the classifiers are unable to detect the Worms attack. This indicates that the UNSW-NB15 is a complex and challenging dataset for multi-class classification with standard ML classifiers.

Table 14: F1-score of standard ML classifiers

	LR	**KNN**	**NB**	**SL**	**SR**	**RF**		**LR**	**KNN**	**NB**	**SL**	**SR**	**RF**
Analysis	**0.29**	0.23	0.08	0.24	0	0.12	Backdoor	0.05	0.14	0.16	0.07	0	0.04
DoS	0.30	0.21	0.11	0.30	**0.48**	0.47	Exploits	0.56	0.49	0.03	0.55	0.53	0.54
Fuzzers	0.54	0.49	0.05	0.52	0.43	0.53	Generic	0.88	0.78	0.67	0.76	0.73	0.71
Normal	0.72	0.73	0.55	0.73	0.65	0.65	Reconn.	0.64	0.65	0.01	0.66	0.36	0.41
Shellcode	0.26	0.33	0.19	0.19	0	0	Worms	0	0.16	0.02	0.22	0	0

LR: Logistic Regression; KNN: K-Nearest Neighbors; NB: Naive Bayes; SL: SVM Linear kernel: SR: SVM RBF kernel: DT: Decision Tree: RF: Random Forest

The F1-score of the proposed neural network architectures as well as of [18] is shown in Table 15, where we observe that DNN_3 architecture performs better than DNN_1 and DNN_2 for most attack categories. This implies that deep learning architectures are suitable for the UNSW-NB15 dataset. To reduce the number of connections relative to the

Table 15: F1-score of neural network architectures

	DNN_1	**DNN_2**	**DNN_3**	**S_CNN**	**D_CNN**	**2DC** [18]		**DNN_1**	**DNN_2**	**DNN_3**	**S_CNN**	**D_CNN**	**2DC** [18]
A	0.28	0.22	0.17	0.22	**0.28**	0	B	0.16	0.21	0.23	0.20	**0.23**	0
D	0.28	0.31	0.24	0.23	**0.46**	0	E	0.57	0.58	0.60	0.57	0.56	**0.62**
F	0.53	0.54	**0.61**	0.54	**0.56**	0.06	G	0.79	0.95	0.96	0.92	0.90	**0.97**
N	0.73	0.73	0.73	0.72	0.73	**0.99**	R	0.69	**0.82**	0.76	0.77	**0.79**	0
S	0.40	0.43	0.42	0.45	**0.48**	0	W	0	0.28	0.22	0.26	**0.37**	0

A: Analysis, B: Backdoor, D: DoS, E: Exploits, F: Fuzzers, G: Generic, N: Normal, R: Reconn., S: Shellcode, W: Worms; FN-1: FFNN with 1 layer; FN-2: FFNN with 2 layers; FN-3; FFNN with 3 layers; S-CNN: Shallow CNN; D-CNN: Deep CNN; 2DC: 2D CNN

DNN architectures, S_CNN and D_CNN are used which also share the weight parameters as shown in Fig. 16. It can be observed from Table 15 that the 2D CNN approach of [18] fails to classify the difficult attack categories and performs well only for the Generic and Normal classes. The proposed D_CNN architecture results in the best performance since it can detect even the difficult attack categories of Analysis, Backdoor, Shellcode, and Worms with significantly improved performance relative to [18]. It can be inferred that the deep learning architectures are suitable for multi-class classification of malicious network traffic for the challenging UNSW-NB15 dataset and the proposed D_CNN architecture outperforms [18] as well as other methods.

7 Conclusions and Future Scope

In this research work, the following contributions are presented for addressing the issues in detecting malicious activities in the network:

- **Comparative analysis of datasets**: We conducted a detailed comparative analysis of the traditional and modern benchmark datasets to determine the most suitable dataset to be used for evaluating the performance of malicious activity detection systems. From our analysis and visualization, we concluded that the modern CICIDS2017 and UNSW-NB15 datasets are appropriate for evaluating the performance of detection systems as these datasets fulfill the characteristics of ideal datasets and they contain complex network traffic data as per recent trends.

- **Unknown attack detection**: A novel multi-level classification method is proposed that can identify the network traffic categories as well as detect unknown attacks via the statistical approach based on the unsupervised Gaussian mixture model. We first proposed a novel sample selection algorithm to select the most representative samples of each traffic category; this helps in reducing the number of traffic samples to be processed in the further pipeline. Following this, by learning the statistical characteristics of each traffic category, the outliers from each Gaussian distribution can be detected and the learned models for each traffic category are cascaded as per the decreasing F1 scores. With this method, even novel or zero-day attacks can be detected if the test traffic sample does not belong to any of the learned Gaussian

probability distributions. The experiments conducted on the CICIDS2017 dataset show that the proposed system is able to outperform the existing state-of-the-art methods available in the literature. For binary classification, the proposed multi-level classification method has an improvement of 17% relative to the baseline method of [25]; for multi-class classification for known traffic categories, the improvement is 18% relative to the baseline method of [**?**], and for unknown attack detection, the F1 score is 87%.

- **Augmentation with Regularized WGAN-LLP for traffic imbalance**: To solve the traffic imbalance problem, we designed a regularization method of Wasserstein Generative Adversarial Network Localized Lipschitz Penalty (WGAN-LLP). The proposed method is shown to generate high-quality synthetic traffic data samples both quantitatively as well as qualitatively over the baseline methods of SMOTE, ROS, and known variants of GAN. The baseline methods fail most of the statistical validity tests and the GAN-based methods perform relatively better, but the proposed WGAN-LLP method passes all five validity tests. Also, the generated samples via WGAN-LLP resemble closely to the original traffic data samples for each traffic category. We also observed that as the balance ratio of the dataset increases, the F1 score of the augmented dataset has an improvement of 22.5% relative to WGAN-GP when the dataset is completely balanced (i.e. $BR = 1$). Further, the classification performance is evaluated via three different strategies of TRTR, TRTS, and TSTR and it is shown that using the TRTS and TSTR strategies, the balanced classification performance improves significantly compared to existing methods as the synthetic data samples are realistic as well as diverse. For binary classification, the proposed method has an improvement of 11% for the F1 score relative to the baseline method of [**?**]; and for multi-class classification, the improvement in F1 score is 2.9% relative to the baseline method of [**?**].

- **Low-frequent attack detection**: For improving the low-frequent attack detection, we designed shallow as well as deep neural network architectures based on fully-connected neural networks and convolutional neural networks. The performance was evaluated on the UNSW-NB15 dataset which is a challenging dataset based on our analysis of scatter plot, t-SNE, and Andrew's curve plot. The experimental results showed that the proposed one-dimensional CNN method performs more effectively than the standard machine learning and artificial neural network classifiers. The proposed method obtains an average F1 score of 50.5% with an improvement of 80% relative to the existing method of 2D CNN [18] having an average F1 score of 26.6%. Specifically, the proposed system is able to improve the detection performance of low-frequent attacks of Shellcode (F1 score of 48%) and Worms (F1 score of 37%), whereas, these two traffic attacks had a low detection rate of 33% and 22%, respectively, in the existing work.

For further work, the concept of One-shot learning [43] can be explored to detect the attack classes having insignificant number of samples during training. This can be further extended to learn zero-day novel attacks using the concept of Zero-shot learning where there are no data samples of the novel attacks available during the training phase. An advantage of using these ideas is that the learning model need not be re-trained again to learn the behavior of any specific network attacks, and also the system can detect novel unknown attacks during the testing phase.

List of Paper Publications

The following papers are published as a part of this research work:

P_1 : R. Chapaneri, and S. Shah, "A Comprehensive Survey of Machine Learning-Based Network Intrusion Detection," ***Smart Innovation, Systems and Technologies, Springer***, vol. 104, pp. 345—356, **Oct 2018**.

https://doi.org/10.1007/978-981-13-1921-1_35 (Scopus Indexed)

P_2 : R. Chapaneri, and S. Shah, "Detection of Malicious Network Traffic using Convolutional Neural Networks," ***Proc. of 10th IEEE Intl. Conf. on Computing, Communication and Networking Technologies*** **(ICCCNT), IIT Kanpur**, **Jul 2019**.

https://doi.org/10.1109/ICCCNT45670.2019.8944814 (Scopus Indexed)

P_3 : R. Chapaneri, and S. Shah, "Multi-level Gaussian Mixture Modeling for Detection of Malicious Network Traffic," ***Journal of Supercomputing, Springer***, vol. 77, no. 5, pp. 4618–4638, **Oct 2020**.

https://doi.org/10.1007/s11227-020-03447-z (Scopus, SCI Indexed, Impact factor: 2.557)

P_4 : R. Chapaneri and S. Shah, "Enhanced Detection of Imbalanced Malicious Network Traffic with Regularized Generative Adversarial Networks", ***Journal of Network and Computer Applications, Elsevier***, vol. 202, article no. 103368, **Mar 2022**.

https://doi.org/10.1016/j.jnca.2022.103368 (Scopus, SCI Indexed, Impact factor: 7.574)

References

[1] Z. Ahmad, K. Shahid, S. Wai, J. Abdullah, and F. Ahmad, " Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol 32, Oct 2020

[2] S. Shah, and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to Snort system," *Future Generation Computer Systems*,vol 80, pp.157–170, Mar 2018

[3] V. Chandola, A. Banerjee, and V.Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41 pp. 1–58, Jul 2009

[4] G. Karatas, O. Demir, and O. Sahingoz, "Increasing the performance of machine learning-based IDSs on an imbalanced and up-to-date dataset," *IEEE Access*, vol. 8, pp. 32150–32162, Feb 2020.

[5] M. Parsaei, S. Rostami, and R. Javidan, "A hybrid data mining approach for intrusion detection on imbalanced NSL-KDD dataset," *Intl. Journal of Advanced Computer Science and Applications*, vol. 7, no. 6, pp. 20–25, Jun 2016.

[6] G. Douzas, F. Bacao, and F. Last, "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE," *Information Sciences*, vol. 465, pp. 1–20, Oct 2018.

[7] H. Zhang, L. Huang, C. Wu, and Z. Li, "An effective convolutional neural network based on SMOTE and Gaussian mixture model for intrusion detection in imbalanced dataset," *Computer Networks*, vol. 177, no. 107315, May 2020.

[8] A. Abdulrahman, and M. Ibrahem, "Toward constructing a balanced intrusion detection dataset," *Samarra Journal of Pure and Applied Science*, vol. 2, no. 3, pp. 132–142, Oct 2020.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, Dec 2014.

[10] X. Zhu, Y. Liu, J. Li, T. Wan, and Z. Qin, "Emotion classification with data augmentation using generative adversarial networks," in *Proc. of Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pp. 349–360, Jun 2018.

[11] Z. Li, T. Zheng, Y. Wang, Z. Cao, Z. Guo, and H. Fu, "A novel method for imbalanced fault diagnosis of rotating machinery based on generative adversarial networks," *IEEE Trans. on Instrumentation and Measurement*, vol. 70, pp. 1–17, Jul 2020.

[12] U. Fiore, A. Santis, F. Perla, P. Zanetti, and F. Palmieri, "Using generative adversarial networks for improving classification effectiveness in credit card fraud detection," *Information Sciences*, vol. 479, pp. 448–455, Apr 2019.

[13] D. Shu, N. Leslie, C. Kamhoua, and C. Tucker, "Generative adversarial attacks against intrusion detection systems using active learning," in *Proc. of the 2nd ACM Workshop on Wireless Security and Machine Learning*, pp. 1–6, Jul 2020.

[14] J. Chen, D. Wu, Y. Zhao, N. Sharma, M. Blumenstein, and S. Yu, "Fooling intrusion detection systems using adversarially autoencoder," *Digital Communications and Networks*, Nov 2020.

[15] M. Shahriar, N. Haque, M. Rahman, and M. Alonso, "G-ids: Generative adversarial networks assisted intrusion detection system," in *Proc. of 44th IEEE Annual Computers, Software, and Applications Conference* (COMPSAC), pp. 376–385, Jul 2020.

[16] S. Huang, and K. Lei, "IGAN-IDS: An imbalanced generative adversarial network towards intrusion detection system in ad-hoc networks," *Ad Hoc Networks*, vol. 105, no. 102177, May 2020.

[17] K. Vigneswaran, R. Vinayakumar, K. Soman and P. Poornachandran, "Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security," in *Proc. of* 9^{th} *IEEE Intl. Conf. on Computing, Communication and Networking* (ICCCNT), Bengaluru, India, Jul 2018.

[18] S. Potluri, S. Ahmed and C. Diedrich, "Convolutional neural networks for multi-class intrusion detection system," in *Proc. of the Intl. Conf. on Mining Intelligence and Knowledge Exploration*, pp. 225-–238, Springer, Cham, Dec 2018.

[19] H. Hindy, C. Tachtatzis, R. Atkinson, E. Bayne, and X. Bellekens, "Developing a Siamese network for intrusion detection systems," *Proc. of the 1st Workshop on Machine Learning and Systems*, pp. 120 – 126, Apr 2021.

[20] A. Shirvai, H. Shirvai, M. Tavallaee and A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," Computers and Security, vol. 31, no. 3, pp. 357-374, 2012.

[21] Tavallaee, Mahbod and Bagheri, Ebrahim and Lu and Wei, "A detailed analysis of the KDD CUP 99 dataset," in Compuational Intelligence for Security and Defense Applications, 2009.

[22] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems," in *Proc. of IEEE Military Communications and Information Systems Conference* (MilCIS), Canberra, Australia, pp. 10–15, Nov 2015.

[23] M. Bhuyan, D. Bhattacharyya and J. Kalita, "Towards Generating Real-life Datasets for Network Intrusion Detection," International Journal of Network Security, vol. 17, pp. 683-701, 2015.

[24] I. Sharafaldin, A. Lashkari, and A. Ghorbani, "A detailed analysis of the CICIDS2017 data set," in *Proc. of Intl. Conf. on Information Systems Security and Privacy*, pp. 172–188, Jan 2018.

[25] D. Pérez, S. Alonso, A. Morán, M. Prada, J. Fuertes, and M. Domínguez, " Comparison of network intrusion detection performance using feature representation," in *Intl. Conf. on Engineering Applications of Neural Networks*, pp. 463–475, May 2019.

[26] N. Marir, H. Wang, G. Feng, B. Li, and M. Jia, "Distributed abnormal behavior detection approach based on deep belief network and ensemble SVM using spark", *IEEE Access*, pp. 59657–59671, Nov 2018

[27] D. Aksu, and M. Aydin, "Detecting port scan attempts with comparative analysis of deep learning and support vector machine algorithms," in *IEEE Intl. Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) Turkey*, pp. 77–80, Dec 2018

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", in *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jan 2014.

[29] F. Chollet, and others, Keras [Internet], GitHub; available from: https://github.com/fchollet/keras, 2015.

[30] D. Kingma, and Jimmy Ba, "Adam: A Method for Stochastic Optimization", in *Proc. of the 3rd International Conference on Learning Representations*, pp. 1–13, May 2015.

[31] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks", in *Intl. Conf. on Machine Learning*, pp. 214–223, Jul 2017.

[32] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs", in *Proc. of the 31st International Conference on Neural Information Processing Systems*, pp. 5769–5779, Dec 2017.

[33] H. Petzka, A. Fischer, and D. Lukovnicov, "On the regularization of Wasserstein GANs", in *Proc. of the 6th Intl. Conf. on Learning Representations*, May 2018.

[34] J. Wu, Z. Huang, J. Thoma, D. Acharya, and L. Gool, "Wasserstein divergence for GANs", in *Proc. of the European Conference on Computer Vision*, pp. 653–668, Sep 2018.

[35] H. Thanh-Tung, T, Tran, and S. Venkatesh, "Improving generalization and stability of generative adversarial networks", in *Proc. of the Intl. Conf. on Learning Representations*, May 2019.

[36] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test", in *Journal of Machine Learning Research*, vol. 13, no. 25, pp. 723–773, Mar 2012.

[37] L. Maaten, and G. Hinton, "Visualizing data using t-SNE", in *Journal of Machine Learning Research*, vol. 9, pp. 2579—2605, Nov 2008.

[38] D. Aksu, and M. Aydin, "Detecting port scan attempts with comparative analysis of deep learning and support vector machine algorithms", in *Proc. of IEEE Intl. Congress on Big Data, Deep Learning and Fighting Cyber Terrorism*", pp. 77–80, Dec 2018.

[39] D. Pérez, S. Alonso, A. Morán, M. Prada, J. Fuertes, and M. Dománguez, "Comparison of network intrusion detection performance using feature representation", in *Proc. of Intl. Conf. on Engineering Applications of Neural Networks*, pp. 463–475, May 2019.

[40] A. Yulianto, P. Sukarno, and N. Suwastika, "Improving AdaBoost-based intrusion detection system performance on CICIDS 2017 dataset", in *Journal of Physics: Conference Series*, vol 1192, no. 1, Mar 2019.

[41] I. Sharafaldin, A. Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", in *Proc. of 4th International Conference on Information Systems Security and Privacy*, pp. 108–116, Jan 2018.

[42] J. Lee, and K. Park, "GAN-based imbalanced data intrusion detection system", in *Personal and Ubiquitous Computing*, vol. 25, no. 1, pp. 121–128, Feb 2021.

[43] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra, "Matching networks for one shot learning", in *Advances in Neural Information Processing Systems*", 2016.